Universidad Autónoma de Madrid

Escuela Politécnica Superior

TRABAJO DE FIN DE MÁSTER

MÉTODOS DE APRENDIZAJE AUTOMÁTICO PARA DETECCIÓN DE ANOMALÍAS

Máster Universitario en Investigación e Innovación en las Tecnologías de la Información y de las Comunicaciones

Autor: Juan Bella Santos
Tutor: José Ramón Dorronsoro Ibero
Grupo de Investigación de Aprendizaje Automático
Departamento de Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid

4 de septiembre de 2019

MÉTODOS DE APRENDIZAJE AUTOMÁTICO PARA DETECCIÓN DE ANOMALÍAS

Autor: Juan Bella Santos
Tutor: José Ramón Dorronsoro Ibero

Grupo de Investigación de Aprendizaje Automático
Departamento de Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid

4 de septiembre de 2019

Agradecimientos

Quiero agradecer al Grupo de Investigación de Aprendizaje Automático la posibilidad de hacer este trabajo con ellos, en especial a Jose, por todo lo que me ha enseñado.

También se lo quiero agradecer a mis padres y mis abuelos, por estar siempre ahí, por ayudarme, por comprenderme, por apoyarme y por quererme en cualquier circunstancia.

Finalmente me gustaría agradecérselo también a mis amigos, a Ada, a Caros y a Jaime, por hacerme pasar buenos ratos, por ayudarme a desconectar y por hacer más llevaderos los momentos de frustración.

Por todo ello y más, muchísimas gracias a todos los que habéis estado a mi alrededor en este último año.

Abstract

Abstract —

In this end-of-master work, different methods are analysed for the detection of outliers or anomalies, i.e. those points found in our data set but which do not comply with the standard. The presence of these points can make the predictions made by a model not as accurate as we would like, so it is necessary to use methodologies to help us identify them. In order to do this, we start from the study of machine learning methods widely known as those of nearest neighbours, Random Forests and support vector machines for the subsequent study of four outlier detection methods.

The methods of anomaly detection to be studied are robust covariance, local outlier factor, isolation forests and one class support vector machines. The first of them tries to find an ellipse containing the points considered as normal. The second of the methods seeks to assign to each of the data a score according to the distance to which their neighbours are, so that the most isolated points are considered as anomalies. The third method to be studied is based on the construction of binary trees with the idea that the outliers are in nodes closer to the root while the normal points are located at greater depth. Finally, the last method treats the problem as if there were only two classes, that of normal data and that of those who are not, so its goal is to find the hyperplane that best separates them.

After this, the different methods are tested from two different perspectives. The first one consists in the use of two synthetic problems in which the effects of the different hyperparameters of the models on the solutions obtained are analysed. The second one is the application of the methods in real datasets, both classification (Oima Inidians diabetes) and regression (Boston Housing and eolic prediction in Australia). In this section we propose an estimator composed of an outlier detection method and a prediction model in order to facilitate the evaluation of the first one in any type of problem, even if we do not know a priori which points are outliers and which are not. In addition to the four methods studied, a voting system has also been applied when classifying the different points as outliers. With all of them, a variable improvement can be observed in the prediction results, both in the classification problem and in the regression ones.

We conclude that, of these methods, the ones that offer the best results are the Local Outlier Factor, the one-class SVM, and the voting system. In addition, possible ways of future development are proposed.

Key words — Outliers, anomaly detection, machine learning.

Resumen

Resumen —

En este trabajo de fin de máster se analizan diferentes métodos para la detección de outliers o anomalías, es decir, aquellos puntos que se encuentran en nuestro conjunto de datos pero que no se ajustan a la norma. La presencia de estos puntos puede hacer que las predicciones realizadas por un modelo no sean tan precisas como nos gustaría, por lo que es necesario el uso de metodologías que nos ayuden a identificarlos. Para ello se va a partir del estudio de métodos de aprendizaje automático ampliamente conocidos como los de vecinos próximos, Random Forests y máquinas de vectores de soporte para el posterior estudio de cuatro métodos de detección de outliers.

Los métodos de detección de anomalías que se van a estudiar son el de covarianza robusta, el del factor de outlier local, el de los bosques de aislamiento y el de las máquinas de vectores de soporte de una clase. El primero de ellos trata de encontrar una elipse que contenga los puntos considerados como normales. El segundo de los métodos busca asignar a cada uno de los datos una puntuación en función de la distancia a la que se encuentran sus vecinos, de tal manera que los puntos más aislados sean considerados como anomalías. El tercer método que se va a estudiar se basa en la construcción de árboles binarios con la idea de que los outliers se encuentren en hojas más cercanas a la raíz mientras que los puntos normales se ubiquen a mayor profundidad. Finalmente, el último método trata el problema como si solo existieran dos clases, la de los datos normales y la de los que no lo son, por lo que su objetivo es encontrar el hiperplano que mejor las separa.

Después de esto, se prueban los diferentes métodos desde dos perspectivas diferentes. La primera consiste en la utilización de dos problemas sintéticos en los que se analizan los efectos de los diferentes hiperparámetros de los modelos en las soluciones obtenidas. La segunda consiste en la aplicación de los métodos en conjuntos de datos reales, tanto de clasificación (diabetes de los indios Pima) como de regresión (viviendas de Boston y predicción eólica en Australia). En este apartado se propone un estimador compuesto por un método de detección de outliers y un modelo de predicción con el fin de facilitar la evaluación del primero en cualquier tipo de problema, incluso si no sabemos a priori qué puntos son outliers y cuáles no. Además de los cuatro métodos estudiados, también se ha aplicado un sistema de votación a la hora de clasificar los diferentes puntos como outliers. Con todos ellos se observan una mejoría variable en los resultados de predicción, tanto en el problema de clasificación, como en los de regresión.

Concluimos que, de estos métodos, los que ofrecen mejores resultados son el del factor de outlier local, el de las máquinas de vectores de soporte de una clase, y el sistema de votación. Se proponen, además, posibles vías de desarrollo futuro.

Palabras clave — Outliers, detección de anomalías, aprendizaje automático.

Índice general

Índice de tablas

Índice de figuras

1 Introducción

Uno de los principales problemas a los que nos podemos enfrentar en el aprendizaje automático son los conjuntos de datos contaminados, es decir, muestras de datos que contienen instancias que se alejan de la norma. Este tipo de puntos, conocidos como outliers o anomalías, son observaciones que se desvían tanto de otras como para despertar sospechas de que hayan sido generadas por un mecanismo diferente [1]. Además, pueden causar que nuestros modelos no sean tan precisos como nos gustaría, por lo que nos puede interesar detectarlos y excluirlos de nuestro modelo.

En el campo de la detección de anomalías, nos podemos encontrar con dos procesos que, aunque muy similares, presentan ciertas diferencias entre sí [2]:

- Detección de anomalías: este caso se da cuando partimos de un conjunto de datos contaminado en el que nos podemos encontrar tanto datos considerados como normales, como datos que provienen de una distribución distinta, es decir, outliers.
- Detección de novedades: en cambio, en este caso partimos de un conjunto de datos libre de outliers, por lo que nuestro objetivo es detectar si las nuevas instancias que nos llegan se comportan de forma normal o si son suficientemente diferentes como para considerar que han sido generados por una distribución diferente.

La detección de anomalías puede ser entendida como una detección de outliers no supervisada, ya que no disponemos de etiquetas que nos indiquen qué es una anomalía y qué no. Por otro lado, la detección de novedades pude verse como una detección de outliers semisupervisada puesto que únicamente tenemos información sobre datos normales o limpios [2].

1.1. Detección de anomalías

Como ya se ha comentado, la detección de anomalías se basa en la identificación de observaciones que no son normales dentro de un conjunto de datos contaminado. Para poder hacer esto,

necesitamos de modelos que nos permitan aprender a separar ambos tipos de datos. Algunos de los modelos propuestos para este fin son:

- Determinante de Covarianza Mínima
- Factor de Outlier Local
- Bosque de aislamiento

A continuación se describen brevemente los diferentes métodos.

Determinante de Covarianza Mínima

Una manera de afrontar la detección de outliers es la suposición de que los datos considerados como normales han sido generados por una distribución conocida. De esta manera, si averiguamos cuál es la distribución de nuestros datos, nos resultará sencillo detectar los outliers, ya que únicamente tendremos que comprobar cómo de probable es que un punto haya sido generado por esta o no.

El modelo del Determinante de Covarianza Mínima o Minimum Covariance Determinant (MCD) trata de buscar la elipse con menor determinante de la matriz de covarianzas de los puntos incluidos en ella [3]. Para decidir qué puntos van a ser incluidos en la elipse, este método utiliza la distancia de Mahalanobis (ver apartado 2.1).

Factor de Outlier Local

Otra forma de enfocar la detección de outliers es mediante la comparación de los diferentes puntos respecto a los que se encuentran a su alrededor. El método de Factor de Outlier Local o Local Outlier Factor (LOF) hace esto mediante el cálculo de una puntuación que nos indica si un punto es más o menos anómalo. Esto lo hace mediante la desviación de la densidad local de un punto con respecto a la de sus vecinos más cercanos. La principal ventaja de este método es la utilización de factores tanto locales como globales, lo que nos permite detectar outliers incluso cuando el conjunto de datos presenta varias subagrupaciones [4].

Bosque de aislamiento

Otro método propuesto para la detección de anomalías es el de bosque de aislamiento o Isolation Forest, que se basa en el funcionamiento de los Random Forests para decidir si un punto es un outlier o no. Para ello, se crean diferentes árboles que van haciendo divisiones sobre las diferentes variables con el objetivo de aislar las instancias de nuestro conjunto de datos. En función de a qué profundidad se queden aislados los diferentes puntos, se les dará una puntuación que se utilizará para decidir si son normales o no. Según este método, los datos normales tienden a llegar a mayores profundidades, mientras que los outliers se quedan más cerca de la raíz del árbol [5].

1.2. Detección de novedades

A la hora de detectar novedades, partimos de un conjunto de datos limpio, por lo que nuestro objetivo es decidir si una nueva observación es suficientemente diferente a las que ya teníamos como para decir que no es normal. A grandes rasgos, estos modelos van a definir una frontera que delimite las observaciones iniciales. De esta manera, cuando nos lleguen nuevas observaciones, la decisión de si es normal o no dependerá de a que lado de la frontera se encuentra. Dentro de estos modelos destacan las máquinas de vectores de soporte de una clase, aunque Local Outlier Factor también puede utilizarse para detectar novedades [2].

Máquinas de vectores de soporte de una clase

Para encontrar la frontera que delimita nuestros datos, este modelo utiliza una versión adaptada de las máquinas de vectores de soporte. Esta adaptación consiste en la adición de un nuevo parámetro ν que representa la probabilidad de encontrar una nueva instancia normal fuera de la frontera calculada [6].

1.3. Objetivo

El objetivo de este Trabajo de Fin de Máster es la utilización de estos métodos de detección de outliers y anomalías sobre diferentes conjuntos de datos, con el fin de mejorar los resultados de predicción, tanto en problemas de clasificación como de regresión.

1.4. Estructura del documento

Para poder cumplir este objetivo, vamos a empezar repasando, en el capítulo 2, diferentes métodos de aprendizaje automático que presentan la base de los algoritmos de detección de anomalías que veremos más adelante. Específicamente, los métodos que vamos a ver son el de vecinos próximos en la sección 2.1, Random Forest en la sección 2.2 y máquinas de vectores de soporte en la sección 2.3.

Como ya hemos comentado, después de esto vamos a estudiar varios métodos de detección de anomalías en el capítulo 3. En concreto, vamos a ver el método de Covarianza Robusta en la sección 3.1, Local Outlier Factor en la sección 3.2, Isolation Forest en la sección 3.3 y One-class SVM en la sección 3.4.

A continuación, en el capítulo 4 vamos a proceder a realizar diferentes experimentos con el fin de comprobar el funcionamiento de los distintos métodos y si, como queremos comprobar, podemos mejorar los resultados de predicción. Para ello, en la sección 4.2 vamos utilizar dos conjuntos de datos sintéticos para comprobar cómo afectan los diferentes parámetros de los modelos a la solución final, y en las secciones 4.4, 4.5 y 4.6 veremos los resultados de estos métodos aplicados a conjuntos de datos reales.

Finalmente, en el capítulo 5 se exponen las conclusiones y posibles líneas de ampliación del trabajo realizado.

2

Métodos de aprendizaje automático

A lo largo de este capítulo vamos a ver los modelos clásicos de aprendizaje automático en los que se basan los métodos de detección de outliers y anomalías que veremos más adelante. Primero, en la sección 2.1 se van a explicar brevemente unos conceptos más sencillos, como las distancias entre puntos de un conjunto de datos y los modelos basados en vecinos próximos, que serán necesarios en los apartados 3.1 y 3.2, donde se estudian métodos de detección de anomalías basados en estos conceptos. A continuación, en la sección 2.2 vamos a ver cómo funcionan los Random Forests, que son la base del apartado 3.3. Finalmente, en la sección 2.3, se hace un repaso de las máquinas de vectores de soporte, que utilizaremos en la sección 3.4.

2.1. Vecinos próximos

Dado un conjunto de datos X, este tipo de modelos usan las observaciones más cercanas en el espacio de entrada a un punto $x \in X$, a las que llamaremos vecinos próximos a x, para obtener una predicción $\hat{Y}$. Esta cercanía entre puntos la calculamos mediante el uso de diferentes métricas, como pueden ser la distancia euclídea o la distancia de Mahalanobis [7].

En primer lugar está la distancia euclídea, que es la distancia más sencilla de todas. Consiste en la longitud de la separación en línea recta entre dos puntos en un espacio euclídeo. Dados dos puntos p y q, ambos en el espacio $\mathbb{R}^d$, la distancia euclídea entre ambos es [7],

$$d(p,q) = d(q,p) = \sum_{i}^{d} \sqrt{(q_i - p_i)^2}.$$

La otra distancia mencionada, la de Mahalanobis, en lugar de medir la separación en el plano, busca cuantificar cómo de parecidos son dos puntos. En este caso, dados dos puntos p y q, ambos en el espacio $\mathbb{R}^n$, la distancia de Mahalanobis entre ambos es [3],

$$d(p,q) = d(q,p) = \sqrt{(p-q)^T \Sigma^{-1} (p-q)},$$

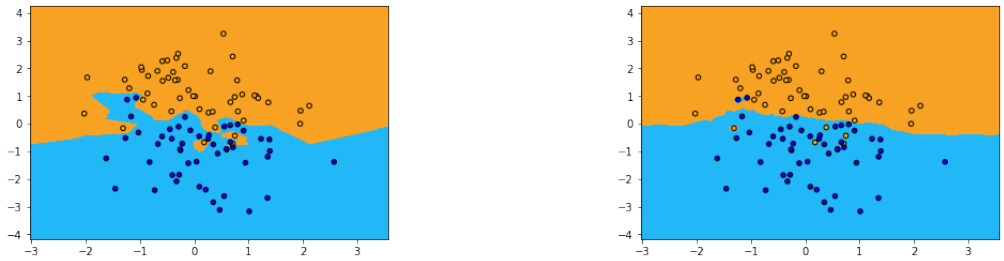

Figura 2.1: Ejemplos de vecinos próximos con 1 vecino (izquierda) y 15 vecinos (derecha).

donde Σ es la matriz de covarianza. Esta matriz debe de ser definida positiva e invertible. Si Σ es la matriz identidad, entonces la distancia de Mahalanobis es igual a la distancia Euclídea.

Como ya hemos comentado, los modelos basados en los vecinos próximos utilizan estas distancias para obtener los puntos más cercanos a x y calcular un valor en función de estos. Para problemas de regresión la solución viene dada por

$$\hat{Y}(x) = \frac{1}{k} \sum_{x_i \in N_k(x)} y_i,$$

donde k es el número de vecinos, $N_k(x)$ es el vecindario de x, definido por los k puntos más cercanos a x en los datos de entrenamiento y y_i el target de los distintos vecinos. Para problemas de clasificación con C clases, la solución viene dada por

$$\hat{Y}(x) = \underset{c \in C}{\operatorname{argmax}} \sum_{x_i \in N_k(x)} \delta(c, y_i),$$

donde $\delta(c, y_i) = 1$ si $c = y_i$ y 0 en caso contrario.

Si escogemos un número de vecinos demasiado pequeño corremos el riesgo de que nuestro modelo se aprenda demasiado los datos de entrenamiento, es decir, estamos corriendo el riesgo de que aparezca overfitting. Podemos ver un ejemplo de este modelo en la figura 2.1. En la figura de la izquierda podemos observar un modelo con $k = 1$; en este caso se está produciendo overfitting, ya que los límites de decisión se ajustan demasiado a los puntos, llegando incluso a dejar a alguno aislado. En la figura de la derecha tenemos un modelo con $k = 15$; en este caso, aunque hay algunos fallos de predicción, la frontera de decisión es más suave, por lo que el error en test será menor.

2.2. Random Forests

En esta sección vamos a ver un modelo de predicción basado en árboles denominado Random Forests, que se basa en la utilización de numerosos árboles, cada uno de ellos dividiendo el espacio muestral en función de ciertos atributos, escogidos de forma aleatoria para cada uno de ellos. Para esto vamos a empezar viendo el funcionamiento de los árboles de clasificación y regresión.

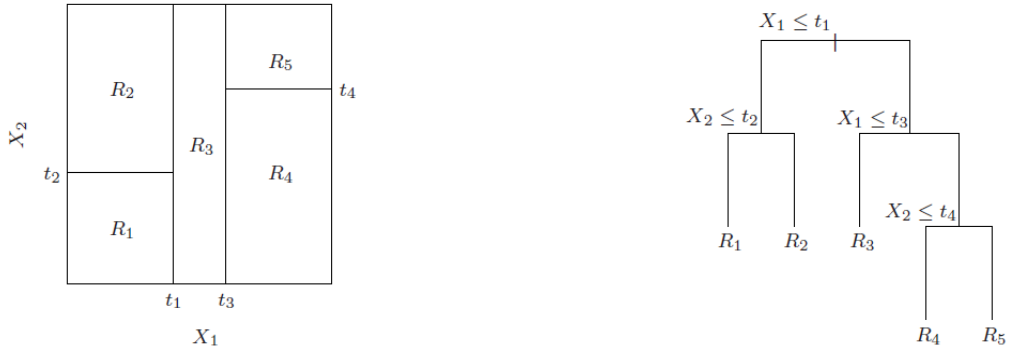

Figura 2.2: Espacio muestral dividido por un árbol (izquierda) y ejemplo de árbol de decisión (derecha).

2.2.1. Árboles de clasificación y regresión

Los métodos basados en árboles tratan de crear un árbol binario que intenta separar el espacio de entrada en un conjunto de rectángulos (figura 2.2, izquierda), para después entrenar un modelo sencillo en cada uno de estos. Para hacer esto, empezando desde el nodo raíz del árbol en el que se encuentran todos los datos, se selecciona un atributo y un valor por el cual partiremos nuestro conjunto de datos, quedando los valores menores o iguales a la izquierda y los mayores a la derecha. Un ejemplo de este tipo de árboles y el resultado en el conjunto de datos lo podemos ver en la figura 2.2, extraída de [7].

La principal ventaja de estos árboles es su interpretabilidad, ya que con un único árbol podemos dividir los datos de una forma clara.

Para la utilización de árboles en problemas de regresión, vamos a suponer que estamos trabajando con un conjunto de N datos de entrada con d dimensiones, $X = \{x_1, x_2, ..., x_n | x_i \in \mathbb{R}^d\}$, y sus targets correspondientes, $Y = \{y_1, y_2, ..., y_n | y_i \in \mathbb{R}\}$. A la hora de construir el árbol, el algoritmo necesita decidir los atributos que se utilizarán para la división de las ramas, los valores de los puntos de división y la topología del árbol. Para esto, lo primero que debemos hacer es definir nuestro modelo, que en el caso de tener M regiones $R_1, R_2, ..., R_M$, quedaría como [7]

$$f(x) = \sum_{m=1}^{M} c_m I(x \in R_m),$$

donde c_m es una constante asociada a cada región e $I(x \in R_m)$ es 1 cuando x se encuentra en R_m y 0 en caso contrario. Para calcular el valor óptimo de $\widehat{c}_m$, podemos utilizar como criterio la suma de cuadrados $\sum(y_i - f(x_i))^2$, lo que nos daría como solución la media de y_i en la región R_m, [7]

$$\widehat{c}_m = \text{ave}(y_i | x_i \in R_m) = \frac{1}{|R_m|} \sum_{x_i \in R_m} y_i,$$

donde $|R_m|$ es el número de puntos que hay en la región R_m

Al igual que podemos utilizar los árboles de decisión para problemas de regresión, también los podemos utilizar para problemas de clasificación. En este caso, las variables de salida en lugar de ser números reales serán los valores de las diferentes clases posibles.

La principal diferencia con los árboles de regresión es la manera de seleccionar el valor de cada una de las regiones R_m. En este caso, primero calculamos la proporción de nodos de cada clase que hay en el nodo m mediante, [7]

$$\hat{p}_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} I(y_i = k),$$

siendo N_m el número de observaciones y k la clase. Una vez hecho esto para cada una de las clases disponibles, le asignamos a dicha región una de estas mediante el criterio

$$k(m) = \text{argmax}_k \, \hat{p}_{mk}.$$

El problema de este tipo de métodos es que seleccionar la mejor forma de dividir los datos es, de forma general, computacionalmente inviable [7]. La forma de solucionar esto, es mediante la utilización de un algoritmo codicioso, es decir, un algoritmo que encuentre una solución global óptima aproximada mediante la búsqueda de soluciones óptimas locales. Para ello, definimos la variable divisoria j y el punto de división s y definimos dos regiones,

$$R_1(j, s) = \{X | X_j \leq s\} \text{ y } R_2(j, s) = \{X | X_j > s\}.$$

Ahora, únicamente tenemos que buscar la variable j y el punto s que resuelven, [7]

$$\min_{j,s} \left[\min_{c_1} \sum_{x_i \in R_1(j,s)} (y_i - c_1)^2 + \min_{c_2} \sum_{x_i \in R_2(j,s)} (y_i - c_2)^2 \right].$$

De esta manera, para unos valores de j y s cualesquiera, la solución en el caso de regresión viene dada por,

$$\hat{c_1} = \text{ave}(y_i | x_i \in R_1(j, s)) \text{ y } \hat{c_2} = \text{ave}(y_i | x_i \in R_2(j, s)).$$

Puesto que, dada una variable divisoria, encontrar el mejor punto de división es un proceso rápido, hallar la mejor pareja (j, s) es computacionalmente viable.

Una vez seleccionado los mejores valores para la división, debemos decidir hasta qué punto vamos a dejar crecer el árbol. Un árbol demasiado grande puede conllevar un riesgo de overfit, mientras que uno pequeño puede hacer que no capturemos bien la estructura de los datos. La altura del árbol es un parámetro de complejidad del modelo, el cual debemos elegir durante la creación del mismo, ya que el valor óptimo depende de los datos [7].

2.2.2. Aleatorización en la construcción de árboles

Un importante problema que presentan los árboles es su gran varianza [7]. Esto se debe a que un error en un punto de división se propaga automáticamente por las divisiones inferiores. Para evitar este inconveniente se pueden utilizar las siguientes técnicas:

- Bagging.
- Selección aleatoria de atributos.

La primera de las técnicas, la de bagging, consiste en crear una familia de árboles T_m, $1 \leq m \leq M$,

a partir de M submuestras del conjunto de datos original X obtenidas mediante bootstraping (generación de nuevos conjuntos utilizando muestreo aleatorio con reemplazamiento) para después promediarlos. Haciendo esto, nos aseguramos de que los árboles están idénticamente distribuidos con un sesgo pequeño [8].

En el caso de que los T_m árboles sean independientes con una varianza común σ^2, entonces la varianza media es σ^2/M. Por otro lado, si la correlación por parejas es $\rho > 0$, la varianza media será [8]

$$\rho\sigma^2 + \frac{1-\rho}{M}\sigma^2.$$

Si aumentamos el valor de M, el segundo término tiende a desaparecer, mientras que el primero se mantiene. Esto hace que la correlación por parejas de los árboles creados por este método limite los beneficios del promediado.

En la segunda de las técnicas, la selección aleatoria de atributos, en lugar de hacer divisiones sobre los d atributos posibles, en cada nodo se selecciona de forma aleatoria un subconjunto de los mismos. Con esto nos aseguramos un ρ pequeño [8]. Normalmente, los valores escogidos para el número de variables escogidas son $\sqrt{d}$, $\lfloor d/3 \rfloor$ o incluso 1. También podemos ver este valor como un posible hiperparámetro de nuestro modelo a seleccionar durante su creación.

Generalmente, cuando el número de variables decrece, el sesgo aumenta [7].

2.2.3. Random Forests

Los Random Forests son una modificación del modelo de bagging que construye una colección de árboles no correlacionados entre sí, para después promediarlos. La idea principal de este método es mejorar la reducción de la varianza que nos proporciona el método de bagging, haciendo más pequeña la correlación entre los árboles, sin, en la medida de lo posible, incrementar la varianza [7]. Esto lo podemos conseguir durante el proceso de creación del árbol, seleccionando de forma aleatoria los diferentes atributos. Cada uno de los árboles que van a formar el bosque lo construimos de la siguiente manera:

1. Obtenemos una submuestra de tamaño N del conjunto de datos X de forma aleatoria y con reemplazo.

2. Si disponemos de d variables de entrada, en cada nodo seleccionaremos de forma aleatoria un conjunto $d' \ll d$ de estas entre las que finalmente elegiremos una para la división de los datos.

3. Hacemos crecer el árbol todo lo posible, sin podarlo.

De forma intuitiva, cuanto menor sea el número de variables de entrada seleccionadas, menor será la correlación entre dos árboles cualquiera del conjunto total, lo que hará que se reduzca la varianza. El número de variables que se suele escoger de forma aleatoria es $\sqrt{d}$, aunque podemos llegar a escoger incluso una única variable [7].

La función de predicción de Random Forest para problemas de regresión es la siguiente,

$$\hat{f}_{rf}^{M}(X) = \frac{1}{M}\sum_{m=1}^{M} T_m(X),$$

donde M es el número de árboles que tenemos en nuestro modelo. En el caso de problemas de clasificación, cada uno de los árboles seleccionará una clase y la que más veces haya salido seleccionada será con la que se clasifique el punto dado.

Una característica importante en Random Forests es la utilización de las muestras "fuera de bolsa" (OOB por sus siglas en inglés) para comprobar el error de predicción. Esto consiste en comprobar el error del punto x_i utilizando para predecir únicamente los árboles en los que no se ha utilizado x_i para construirlos. Esta medida del error es equivalente al error calculado mediante validación cruzada [7].

Otra característica destacable que nos ofrecen los Random Forests es la capacidad de utilizarlos para calcular cómo de importantes son las diferentes variables de nuestro conjunto de datos. Esto lo hacemos prediciendo en un árbol con las muestras fuera de bolsa y calculando la precisión de las predicciones. Después intercambiamos los valores de una de las variables de forma aleatoria entre las diferentes muestras fuera de bolsa y volvemos a predecir y calcular la precisión. La diferencia entre estas precisiones es la que, promediada entre todos los árboles, utilizamos como medida de importancia.

Una cosa a tener en cuenta a la hora de utilizar este modelo es la cantidad de hiperparámetros que tiene, lo que dificulta la búsqueda de los parámetros óptimos. A continuación se explican los diferentes hiperparámetros que nos podemos encontrar para Random Forests en `scikit-learn`, una de las principales librerías de aprendizaje automático para `Python` [9]

- Número de estimadores: es la cantidad de árboles que van a formar el modelo.
- Criterio: es la función que se va a utilizar para evaluar la calidad de la división realizada en cada nodo.
- Profundidad: nos permite decidir cuánto vamos a dejar crecer los árboles.
- Número mínimo de muestras por división: indica la cantidad de muestras necesarias para poder dividir un nodo.
- Número mínimo de muestras por hoja: indica la cantidad de puntos que debe tener un nodo para ser considerado hoja.
- Número máximo de variables: es la cantidad de variables a tener en cuenta a la hora de dividir un nodo.
- Valor mínimo de decrecimiento de la impureza: permite controlar si un nodo se divide o no en función de la mejoría que nos aporta.

2.3. Máquinas de vectores de soporte (SVM)

Este modelo se basa en la utilización de hiperplanos para separar las diferentes clases de los datos. Aunque nos vamos a centrar en su uso para problemas de clasificación, también se pueden utilizar para problemas de regresión.

A lo largo de esta sección primero se abordarán los problemas de clasificación cuyas clases se pueden separar de forma lineal (sección 2.3.1), después se tratarán los problemas en los que las clases pueden estar solapadas (sección 2.3.2) y finalmente se verá como cómo podemos aplicar kernels para mejorar el funcionamiento de estos modelos (sección 2.3.3).

2.3.1. Clasificación de problemas linealmente separables

La idea principal de las SVM's para clasificación (SVC) es la de encontrar el hiperplano que separa las diferentes clases manteniendo el mayor espacio posible entre ellas y el menor entre cada clase y el hiperplano.

Si partimos de un conjunto de datos $X = x_1, ..., x_n$ de dimensión d con unas etiquetas de clases asociadas $Y = y_1, ..., y_n$ dónde $y_i = \pm 1$ y un hiperplano $\pi : w \cdot x + b = 0$ en el que w es ortogonal a π y si suponemos que w "apunta" hacia los puntos con $y_i = 1$, podemos definir la distancia entre un punto x y π como

$$d(x_i, \pi) = \frac{|w \cdot x_i + b|}{||w||} = \frac{y_i(w \cdot x_i + b)}{||w||}.$$

El margen entre cada punto y el hiperplano es la mínima distancia entre ambos, por lo que, si trabajamos con el plano normalizado tal que $\min_i y_i(w \cdot x_i + b) = 1$, podemos definir esta distancia como

$$\min_i d(x, \pi) = \min_i \frac{y_i(w \cdot x_i + b)}{||w||} = \frac{1}{||w||}.$$

Como el objetivo inicial era separar lo máximo posible las clases, nuestro problema a resolver es el siguiente.

$$\max_{w,b} f(w, b) = \frac{1}{||w||} \equiv \min_{w,b} f(w, b) = ||w|| \equiv \min_{w,b} f(w, b) = \frac{1}{2}||w||^2 \quad \text{sujeto a } y_i(w \cdot x + b) \geq 1$$

Este problema es conocido como el problema primal. Aunque se trata de un problema aparentemente sencillo, estamos ante un problema de optimización convexa con un alto número de restricciones, por lo que en lugar de intentar minimizarlo directamente, es más conveniente usar multiplicadores de Lagrange para obtener el problema dual [10].

Para este problema, el Lagrangiano es

$$L(w, b, \alpha) = \frac{1}{2}||w||^2 - \sum_i \alpha_i(y_i(w \cdot x_i + b) - 1),$$

con $\alpha_i \geq 0$. Para obtener el problema dual, primero debemos minimizar $L(w, b, \alpha)$ respecto a w y b. Para ello obtenemos $\nabla_w L = 0$ y $\frac{\partial L}{\partial b} = 0$. De aquí obtenemos que

$$w = \sum_{i=1}^{N} \alpha_i y_i x_i, \qquad \sum_{i=1}^{N} \alpha_i y_i = 0.$$

Si sustituimos estos resultados en $L(w, b, \alpha)$ llegamos a la función dual, que queda como

$$\begin{aligned} \Theta(\alpha) &= \frac{1}{2}||w||^2 - \sum_{i=1}^{N} \alpha_i y_i x_i w - \sum_{i=1}^{N} \alpha_i y_i b + \sum_{i=1}^{N} \alpha_i \\ &= \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{k=1}^{N} \alpha_i \alpha_k y_i y_k x_i x_k \end{aligned}$$

sujeto a $\alpha_i \geq 0$ y $\sum_{i=1}^{N} \alpha_i y_i = 0$.

Puesto que por construcción $\Theta(\alpha) \leq L(w, b, \alpha) \leq f(w, b)$, nuestro problema a resolver ahora es

$$\max_{\alpha} \Theta(\alpha)$$

y se deben cumplir las restricciones $w^* = \sum_{i=1}^{N} \alpha_i^* y_i x_i, \sum_{i=1}^{N} \alpha_i^* y_i = 0$ y $\alpha_i \geq 0$. Además, se deben cumplir las condiciones de Karush-Kuhn-Tucker (KKT) [7],

$$y_i(w^* \cdot x_i + b^*) - 1 \geq 0,$$

$$\alpha_i^* \geq 0,$$

$$\alpha_i^*[y_i(w^* \cdot x_i + b^*) - 1] = 0 \ \ \forall i.$$

Si $\alpha_i^* > 0$, entonces $y_i(w^* \cdot x_i + b^*) = 1$, y x_i se encuentra en el límite del margen. Estos puntos son conocidos como vectores de soporte. En cambio, si $y_i(w^* \cdot x_i + b^*) > 1$, x_i no se encuentra sobre el borde, y $\alpha_i^* = 0$.

El valor de b^* lo podemos obtener resolviendo $\alpha_i^*[y_i(w^* \cdot x_i + b^*) - 1] = 0$ para cualquiera de los puntos de soporte. Haciendo esto se tiene:

$$b^* = \frac{1 - y_i x_i^T w^*}{y_i} = y_i - x_i^T w^*.$$

En la práctica es mejor hacer la media de los valores de b para todo $\alpha_i^* > 0$

$$b^* = \frac{1}{N_S} \sum_{\alpha_i^* > 0} (y_i - x_i^T w^*),$$

siendo N_S el número de vectores de soporte.

2.3.2. Clasificación de problemas linealmente no separables

En la mayoría de los casos, los datos no son linealmente separables, es decir, las clases se solapan. Para resolver este problema, lo que debemos hacer es modificar el modelo de tal manera que permitamos a ciertos puntos estar en el lado "incorrecto" del hiperplano de máximo margen pero con una penalización que aumente con la distancia al mismo.

Para realizar esto debemos introducir al modelo una variable de holgura o slack, $\xi_i \geq 0$, por cada punto en el conjunto de datos. Estas variables hacen que los puntos que están en el lado "correcto" del margen o sobre el mismo tengan $\xi_i = 0$. El resto de puntos tendrán $\xi_i = |y_i - f(x_i)|$.

Tras añadir estas nuevas variables, el problema primal queda de la siguiente manera:

$$\min_{w,b} \frac{1}{2}||w||^2 + C \sum_{i=1}^{N} \xi_i \tag{2.1}$$

sujeto a $y_i(w \cdot x_i + b) \geq 1 - \xi_i$ y $\xi_i \geq 0$ con $i = 1, ..., N$. La variable C se encarga de compensar los valores del margen y la penalización de las variables slack. Por ello, una C pequeña conlleva márgenes amplios y un posible underfit y una C grande, márgenes estrechos y un posible overfit [7].

Para este problema, el Lagrangiano es

$$L(w, b, \xi, \alpha, \beta) = \frac{1}{2}||w||^2 + C\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N}\alpha_i[y_i(w \cdot x_i + b) - 1 + \xi_i] - \sum_{i=1}^{N}\beta_i\xi_i,$$

con $\alpha_i, \beta_i \geq 0$. Al igual que en el problema anterior, debemos calcular $\nabla w = 0$, $\frac{\partial L}{\partial b} = 0$, $\frac{\partial L}{\partial \xi_i} = 0$. De aquí obtenemos que

$$w = \sum_{i=1}^{N}\alpha_i y_i x_i,$$

$$C = \alpha_i + \beta_i,$$

$$\sum_{i=1}^{N}\alpha_i y_i = 0.$$

Con estos datos podemos obtener el problema dual sustituyéndolos en $L(w, b, \xi, \alpha, \beta)$, quedándonos

$$\Theta(\alpha, \beta) = \sum_{i=1}^{N}\alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j x_i x_j,$$

sujeto a $\sum_i \alpha_i y_i = 0, \alpha_i \geq 0, \beta_i \geq 0, \alpha_i + \beta_i = C$. Como β no aparece en Θ, podemos reducir las restricciones asociadas a α y β a $0 \leq \alpha_i \leq C$.

En esta ocasión, las condiciones KKT son

$$y_i(w^* \cdot x_i + b^*) - 1 + \xi_i^* \geq 0,$$

$$\alpha_i^* \geq 0,$$

$$\alpha_i^*[y_i(w^* \cdot x_i + b^*) - 1 + \xi_i^*] = 0,$$

$$\beta_i^*\xi_i^* = 0 \quad \forall i,$$

$$w^* = \sum_{i=1}^{N}\alpha_i^* y_i x_i.$$

El valor de b^* lo podemos obtener resolviendo $\alpha_i^*[y_i(w^* \cdot x_i + b^*) - 1 + \xi_i^*] = 0$ para cualquiera de los puntos de soporte, lo que nos da

$$b^* = y_i - w^* \cdot x_i - \xi_i^*.$$

Finalmente, la función de decisión quedaría de la siguiente manera [7],

$$f(x) = \text{sign}(b^* + w^* \cdot x) = \text{sign}(b^* + \sum_{i=1}^{N}\alpha_i^* y_i x_i x) \tag{2.2}$$

A la hora de entrenar las SVMs, ya sean para clasificación o para regresión, tenemos a nuestra disposición numerosos algoritmos. De todos los existentes, el más utilizado es el de optimización mínima secuencial (SMO). En este algoritmo se trata de encontrar los valores óptimos para las diferentes α de manera iterativa. En la sección 3.4.3 veremos detalladamente cómo aplicar este algoritmo al modelo de One-class SVM.

2.3.3. Utilización de kernels

Según el teorema de Cover [11], un problema de clasificación proyectado a un espacio dimensional mayor es más probable que sea linealmente separable. Para una muestra S con dimensión d y tamaño n, en posición general, el número de problemas linealmente separables es

$$L(n,d) = \begin{cases} 2^n, & \text{si } n \leq d+1 \\ 2\sum_{i=0}^{D} \binom{n-1}{i}, & \text{si } n \geq d+1. \end{cases}$$

Partiendo de esta idea, podemos incluir proyecciones no lineales a las SVMs para poder encontrar así el hiperplano que mejor separa nuestros datos. El problema de proyectar a una dimensión alta es que presenta dos desventajas:

1. La maldición de la dimensionaldad, por el cual necesitamos más patrones para entrenar el modelo. También hace que exista un mayor riesgo de overfit.

2. Un gran aumento del coste computacional.

La primera de estas situaciones, las SVMs la solucionan mediante la maximización del margen, ya que la complejidad del modelo depende solo de éste, no de la dimensión. Para solventar la segunda desventaja, podemos utilizar kernels, para así hacer las proyecciones únicamente de manera implícita.

La idea principal es aumentar la dimensión de nuestros datos yendo de $x \in \mathbb{R}^d$ a $\Phi(x) \in \mathbb{R}^{d'}$ con $d' \gg d$, sin tener que hacer los cálculos de forma explícita. Para ello, como en las SVMs solo tenemos que calcular productos escalares, podemos hacer la transformación mediante una función kernel,

$$k(x_i, x_j) = \Phi(x_i) \cdot \Phi(x_j)$$

Por ejemplo, el kernel polinómico $k(x_i, x_j) = (x_i \cdot x_j + 1)^2$ con $x_i, x_j \in \mathbb{R}^2$, puede ser expresado como

$$\begin{aligned} k(x_i, x_j) &= (x_{i1}x_{j1} + x_{i2}x_{j2} + 1)^2 \\ &= (x_{i1}x_{j1})^2 + (x_{i2}x_{j2})^2 + 2x_{i1}x_{j1}x_{i2}x_{j2} + 2x_{i1}x_{j1} + 2x_{i2}x_{j2} + 1 \\ &= (x_{i1}^2, x_{i2}^2, \sqrt{2}x_{i1}x_{i2}, \sqrt{2}x_{i1}, \sqrt{2}x_{i2}, 1)(x_{j1}^2, x_{j2}^2, \sqrt{2}x_{j1}x_{j2}, \sqrt{2}x_{j1}, \sqrt{2}x_{j2}, 1) \\ &= \Phi(x_i) \cdot \Phi(x_j). \end{aligned}$$

La transformación asociada a este kernel es $\Phi(x) = (x_1^2, x_2^2, \sqrt{2}x_1x_2, \sqrt{2}x_1, \sqrt{2}x_2, 1)$, que lleva los datos de una dimensión 2 a una dimensión 6.

Otro kernel que podemos utilizar es el Gaussiano, $k(x_i, x_j) = \mathrm{e}^{-\gamma||x_i - x_j||^2}$. Su principal ventaja es que la transformación asociada $\Phi(x)$ tiene dimensión infinita, por lo que el teorema de Cover ya no representa una limitación. Al utilizar este kernel, debemos tener en cuenta que aunque nos permite ir a una dimensión mayor, sigue existiendo un posible riesgo de overfit. Este se da cuando el valor de γ es muy alto, ya que puede derivar en gaussianas muy agudas y posiblemente en una gaussiana por cada punto x_i [7].

De esta manera, si utilizamos un kernel para proyectar nuestros datos a una dimensión mayor mediante una transformación $\Phi(x)$ la función de decisión de un problema de clasificación (2.2) quedaría

$$f(x) = \mathrm{sign}(b^* + \sum_{i=1}^{N} \alpha_i^* y_i \Phi(x_i)\Phi(x)) = \mathrm{sign}(b^* + \sum_{i=1}^{N} \alpha_i^* y_i k(x_i, x)).$$

3

Métodos de detección de outliers y anomalías

Como ya hemos contado anteriormente, en función de en qué momento estemos detectando las anomalías podemos diferenciar dos tipos de problemas. El primero consiste en separar los datos normales de los outliers en un conjunto de datos. El segundo problema consiste en entrenar un modelo a partir de una muestra de datos que únicamente contiene datos normales para después identificar posibles puntos anómalos, también llamados novedades.

Para cada uno de estos problemas hay métodos específicos [2]. A lo largo de esta sección vamos a ver tres de ellos para la detección de outliers (Covarianza Robusta en la sección 3.1, Factor de Outlier Local en la sección 3.2 e Isolation Forest en la sección 3.3) y uno para la detección de novedades (One-class SVM en la sección 3.4).

Aunque la clasificación de métodos a seguir durante este capítulo es la mencionada anteriormente hay que destacar que también existen otras formas de dividir los diferentes métodos de detección de outliers. Una de estas formas alternativas de clasificación la podemos encontrar en [12], donde se separan los métodos en función de su funcionamiento. Nos ofrecen cuatro categorías:

- Modelos lineales.
- Modelos de proximidad.
- Agrupaciones de métodos.
- Redes neuronales.

Los métodos de Covarianza Robusta (sección 3.1) y One-class SVM (sección 3.4) pertenecen a la categoría de modelos lineales, el de Factor de Outlier Local (sección 3.2) a la de modelos de proximidad y el de Isolation Forest (sección 3.3) a la de agrupaciones de métodos. Respecto a la categoría de redes neuronales, podemos encontrar una comparación de algunos métodos de este tipo en [13].

3.1. Covarianza Robusta

Una de las dificultades a la hora de detectar outliers en conjuntos de datos con dimensión $d > 3$ es que no podemos apoyarnos en una inspección visual. Una forma de abordar este problema es la utilización de estimadores robustos para la localización y dispersión de distribuciones multivariantes. Uno de los estimadores que hacen esto es el del elipsoide de volumen mínimo (MVE).

Este método lo que busca es el elipsoide de volumen mínimo que contiene h puntos, donde $n/2 \leq h < n$, con n siendo el número de observaciones. El algoritmo que reproduce esta metodología se llama MINVOL, que empieza con un subconjunto de $d+1$ datos y calcula su media y su matriz de covarianza. El elipsoide correspondiente se va agrandando o empequeñeciendo hasta que contenga h puntos. Esto se repite varias veces y se escoge el elipsoide con un volumen menor [3].

Otro método similar es el del mínimo determinante de la covarianza (MCD). En este caso, lo que buscamos son las h observaciones de la muestra cuya matriz de covarianzas tiene el menor determinante [3].

Según [3], el método de MCD presenta varias ventajas sobre el de MVE, por ejemplo:

- Su eficiencia estadística es mejor.
- Presenta una mayor precisión.
- Los estimadores obtenidos de MCD son más robustos y precisos que los obtenidos por MVE, lo que hace que sea mejor a la hora de detectar outliers.

El principal problema de MCD es que no es sencillo de calcular. Como solución a esto, en [3] se propone un nuevo algoritmo basado en MCD y que supera en velocidad a los basados en MVE.

3.1.1. Teorema básico y el C-step

Para conseguir unos estimadores más robustos, debemos partir de la idea de que, desde una aproximación de MCD, es posible hallar otra con un determinante menor. Para ello, primero debemos considerar un conjunto de datos $X = \{x_1, ..., x_n\}$ de d dimensiones. También tenemos que definir $H_1 \subset X$ con $|H_1| = h$ y $h \leq n$. Por defecto tomamos un valor de $h = [(n + d + 1)/2]$. Para este subconjunto H_1 con media $M_1 := \frac{1}{h} \sum_{i \in H_1} x_i$ y matriz de covarianzas

$$S_1 := \frac{1}{h} \sum_{i \in H_1} (x_i - M_1)(x_i - M_1)^t$$

con $\det(S_1) \neq 0$, podemos calcular la distancias relativas de cada punto mediante la distancia de Mahalanobis [3] (ver sección 2.1),

$$d_1(x_i) = \sqrt{(x_i - M_1)^t S_1^{-1} (x_i - M_1)} \quad \text{para } i = 1, ..., n.$$

Teorema. *Si obtenemos un subconjunto H_2 de tal manera que $\{d_1(x_i) : i \in H_2\} := \{d_1(x_1), ..., d_1(x_h)\}$ donde $d_1(x_1) \leq d_1(x_2) \leq ... \leq d_1(x_n)$ son las distancias relativas ordenadas*

y calculamos M_2 y S_2 para este subconjunto, tenemos que [3]

$$det(S_2) \leq det(S_1).$$

Esta igualdad se da únicamente cuando $M_2 = M_1$ y $S_2 = S_1$.

A partir de este teorema, de una forma más genérica y orientado a obtener un algoritmo de ello, podemos describir un C-step en una iteración t como:

- Para un subconjunto H_t con h elementos y los estimadores (M_t, S_t), calculamos las distancias $d_t(x_i)$ para $i = 1, ..., n$, con $h \leq n$.
- Ordenamos las distancias de menor a mayor, lo que da como resultado una permutación π de tal manera que,
$$d_t(\pi(x_1)) \leq d_t(\pi(x_2)) \leq ... \leq d_t(\pi(x_n)).$$
- Creamos un nuevo subconjunto $H_{t+1} := \{\pi(x_1), \pi(x_2), ..., \pi(x_h)\}$.
- Calculamos $M_{t+1} := \text{ave}(H_{t+1})$ y $S_{t+1} := \text{cov}(H_{t+1})$, donde $\text{ave}(x)$ es la media de x y $\text{cov}(x)$ es la matriz de covarianzas de x.

Si repetimos esto varias veces obtenemos un proceso iterativo, en el cuál pararemos si $\det(S_t) = 0$ o si $\det(S_t) = \det(S_{t-1})$. La secuencia

$$... \geq \det(S_{t-1}) \geq \det(S_t) \geq \det(S_{t+1}) \geq ...$$

es siempre positiva y por tanto debe converger. Además, como hay un número finito de subconjuntos de h elementos, debe existir un índice m para el cual $\det(S_m) = 0$ o $\det(S_m) = \det(S_{m-1})$, en el cual el algoritmo habrá convergido.

Corolario. *El subconjunto H de X obtenido de la convergencia del algoritmo de MCD, está separado del subconjunto $X \setminus H$ por un elipsoide [3].*

Todo este proceso descrito no es suficiente para asegurar que el determinante de S_m defina el elipsoide con menor determinante de todos los posibles, aunque sí que es una condición necesaria. De todas maneras, el procedimiento descrito nos ofrece una idea inicial para un algoritmo [3]:

- Escoger varios conjuntos iniciales H_1.
- Aplicar el proceso de C-step a cada uno de ellos hasta que converjan.
- Quedarnos con la solución con un determinante menor.

3.1.2. Construcción del nuevo algoritmo

Partiendo de la idea anterior y añadiendo algunas funcionalidades al proceso de selección de la solución, vamos a poder obtener el algoritmo que estábamos buscando, una versión de MCD fácilmente computable. Los nuevos aspectos que le vamos a añadir al C-step son:

- Crear varios subconjuntos iniciales H_1 a los que aplicar C-steps.

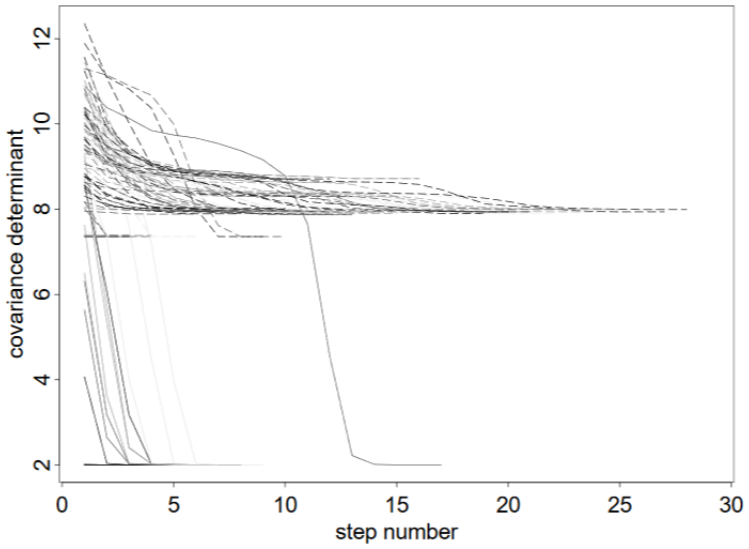

Figura 3.1: Evolución del determinante de la matriz de covarianzas.

- De todos los subconjuntos iniciales, quedarnos únicamente con los que nos vaya a ofrecer mejores resultados, es decir, los que tienen un determinante menor.
- Ir obteniendo soluciones parciales para evitar una sobrecarga de trabajo en muestras grandes.

Creación de varios subconjuntos iniciales

A la hora de crear cada uno de los subconjuntos H_1 seguimos el siguiente procedimiento para cada uno de ellos:

- Creamos un conjunto inicial J que contenga $d+1$ muestras seleccionadas de forma aleatoria.
- Calculamos $M_0 := \text{ave}(J)$ y $S_0 := \text{cov}(J)$.
- En el caso de que $\det(S_0) = 0$, entonces añadimos otra muestra aleatoria y repetimos el paso anterior hasta que $\det(S_0) > 0$.
- Calculamos las distancias de los puntos $d_0^2(x_i) := (x_i - M_0)^t S_0^{-1}(x_i - M_0)$ para $i = 1, ..., k$, siendo k el número de datos en J.
- Ordenamos las distancias de forma ascendente de tal manera que $d_0(\pi(1)) \leq ... \leq d_0(\pi(k))$.
- Creamos un conjunto inicial $H_1 := \{\pi(1), ..., \pi(h)\}$.

Selección de conjuntos iniciales

Si para cada uno de los conjuntos iniciales calculamos las medias, los determinantes y las matrices de covarianza de los pasos intermedios hasta que C-step converja, nos podemos encontrar con un uso excesivo de memoria y coste computacional. Para evitar esto necesitamos poder identificar

rápidamente qué conjuntos iniciales nos van a ofrecer una solución buena y cuales no. En el artículo original [3] nos proponen ejecutar 2 C-steps para cada uno de los subconjuntos iniciales y quedarnos con los 10 que tengan un determinante menor para continuar iterando. Esto se debe a que el valor del determinante de los subconjuntos que nos van a ofrecer una buena solución desciende más rápido que el de aquellos que no, como podemos ver en la figura 3.1, extraída de [3].

Soluciones parciales

El algoritmo descrito, si lo utilizamos en conjuntos de datos pequeños se completa en poco tiempo, pero para conjuntos de datos demasiado grandes el tiempo de computación se ve incrementado [3]. Para solucionar esto, podemos calcular soluciones parciales para subconjuntos que no se solapen para posteriormente comprobar si estas soluciones son apropiadas para el conjunto original.

En el artículo original [3] nos ofrecen estas opciones de divisiones:

- Si $n \leq 600$: aplicaremos el algoritmo sin realizar ninguna división en la muestra.
- Si $600 < n < 1500$: dividiremos los datos como mucho en 4 subconjuntos de 300 o más datos, de tal manera que todos tengan un tamaño similar.
- Si $n \approx 1500$: crearemos 5 subconjuntos de alrededor de 300 muestras.
- Si $n \gg 1500$: dividiremos el conjunto de datos en tantos subconjuntos de 1500 datos como sea necesario y para cada uno de ellos crearemos 5 subconjuntos de unos 300 datos cada uno.

En los casos en los que $n > 600$, en cada uno de los subconjuntos de aproximadamente 300 elementos, ejecutaremos 2 C-steps y seleccionaremos las 10 mejores soluciones para cada uno. A continuación, juntaremos los distintos elementos hasta conseguir un conjunto de 1500 datos como mucho y continuaremos ejecutando C-steps sobre las diferentes soluciones hasta que converjan. En el caso de tener varios subconjuntos de 1500 elementos, se seleccionarán las 10 mejores soluciones de cada uno de ellos y se volverán a ejecutar C-steps sobre la muestra completa hasta llegar a una solución definitiva.

3.1.3. Algoritmo FAST-MCD

Si juntamos los pasos descritos anteriormente, el pseudocódigo del nuevo algoritmo es el siguiente:

- Inicializamos el valor de h con su valor por defecto ($[(n+d+1)/2]$) o con un valor arbitrario que cumpla $[(n+d+1)/2] \leq h \leq n$, donde n es el tamaño de la muestra y d la dimensión.
- En el caso de que $h = n$, M es la media de todo el conjunto de datos y S su matriz de covarianzas.
- Si $n \leq 600$, construimos varios H_1 y claculamos C-steps hasta que el algoritmo converja y obetener la solución (M, S) con un valor menor para $det(S)$.
- Si $n > 600$, construimos los subconjuntos descritos anteriormente hasta obtener una solución (M_{full}, S_{full}) para el conjunto de datos completo.

- Para que nuestros resultados sean más robustos en el caso de provenir de una distribución normal multivariante, realizamos el siguiente ajuste [3],

$$M_{MCD} = M_{full},$$

$$S_{MCD} = \frac{med_i d^2_{full}(x_i)}{\chi^2_{p,0,5}} S_{full}.$$

- Finalmente, podemos aplicar un último paso de ponderación mediante las fórmulas [3]

$$M_{MCD} = \frac{\sum_{i=1}^{n} w_i x_i}{\sum_{i=1}^{n} w_i},$$

$$S_{MCD} = \frac{\sum_{i=1}^{n} w_i (x_i - M_1)(x_i - M_1)^t}{\sum_{i=1}^{n} w_i - 1},$$

donde

$$w_i = \begin{cases} 1 & \text{si } d_{MCD}(x_i) \leq \sqrt{\chi^2_{p,0,975}} \\ 0 & \text{en caso contrario.} \end{cases}$$

3.1.4. FAST-MCD en scikit-learn

A la hora de utilizar este algoritmo en `scikit-learn`, nos hemos dado cuenta de que en la implementación de FAST-MCD tiene ligeras diferencias con la propuesta descrita anteriormente.

- El valor por defecto de h (`support_fraction` en `scikit-learn`) es el mismo que en el algoritmo `ceil(0.5*(n+d+1))`, pero si queremos modificarlo, en lugar de dar un valor fijo, debemos dar un porcentaje de los datos que queremos utilizar.
- En el caso de tener muchos datos y tener que dividirlos en subconjuntos, en lugar de hacer las divisiones en 600 y 1500, las hace en 500 y 1500, no pudiendo modificar estas variables.
- Algo similar pasa con el número de subconjuntos iniciales que se crean, aunque `scikit-learn` realiza este proceso, no podemos indicar cuántos subsets vamos a crear. Por defecto este número oscila entre 10 y 500, dependiendo del tamaño del conjunto de datos.
- Otro aspecto con el que contamos en `scikit-learn` es el método `reweight_covariance(data)`, que calcula la versión ponderada de la solución.

3.1.5. Situaciones de ajuste exacto

En algunas ocasiones nos podemos encontrar con que h o más observaciones se encuentran sobre un mismo hiperplano. Esto se da cuando $\det(S_t) = 0$. En estos casos, la solución que obtenemos es la ecuación del hiperplano en el que se encuentran los datos [3].

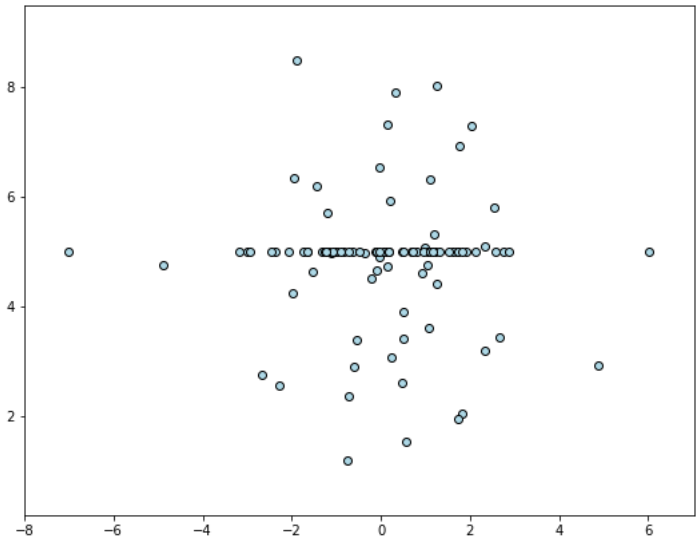

Figura 3.2: Situación de ajuste exacto.

Para comprobar si nos encontramos en una de estas situaciones, el algoritmo realiza los siguientes pasos:

- Si en alguno de los subconjuntos llegamos a una solución (M_{sub}, S_{sub}) con $\det(S_{sub}) = 0$, comprobamos si h o más puntos del conjunto de datos completo se encuentran en este hiperplano y en caso de que sí pase, calculamos (M_{full}, S_{full}) y detenemos el algoritmo.
- En caso contrario, puesto que esta solución se va a encontrar entre las 10 con menor determinante, creamos un nuevo subconjunto H_1' que contenga los h puntos con menor distancia ortogonal al hiperplano y continuamos de forma normal.

En la figura 3.2 podemos ver un ejemplo de esta situación.

3.2. Local Outlier Factor

En ocasiones, nos podemos encontrar con situaciones en las que definir un punto como outlier no sea una decisión binaria. En estos casos nos interesa tener una medida que nos indique cómo de diferente es un punto, de tal manera que podamos decidir a partir de qué momento un punto empieza a ser considerado anómalo.

Para hacer esto, podemos mirar la definición de outlier desde un punto de vista más cercano al concepto de agrupación de datos o cluster. En función de esto, una anomalía es un punto que no pertenece a ninguno de los clusters presentes en nuestro conjunto de datos. El método de Local Outlier Factor propuesto por Breuing [4] utiliza estas ideas para otorgar a cada instancia una puntuación en función de sus vecinos más cercanos.

Una definición de outlier desde este punto de vista puede ser la $DB(pct, dmin)-$outlier, que dice que un punto x_i en una muestra X es un outlier si al menos un porcentaje pct de los objetos en X se encuentran a una distancia superior a $dmin$ de x_i [4]. El problema de esta definición es que solo hace referencia a outliers globales. Por ejemplo, en la figura 3.3, no hay forma de hacer

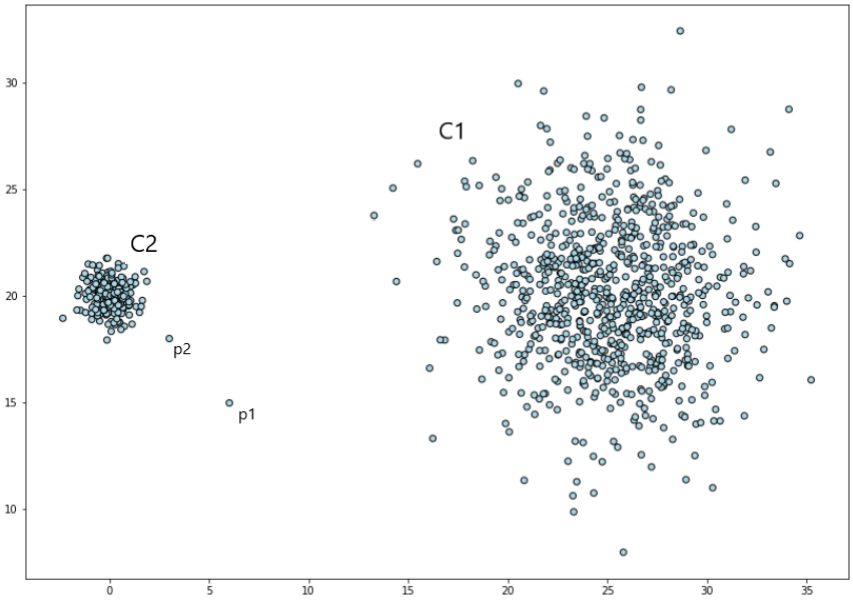

Figura 3.3: Ejemplo de outlier con respecto a varios clusters.

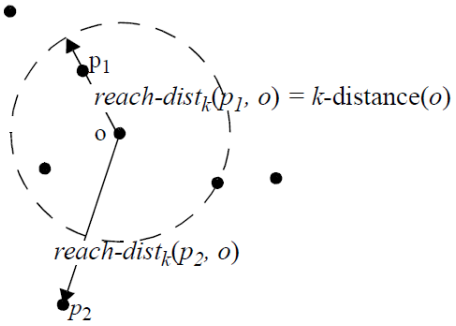

Figura 3.4: Ejemplo de la distancia de alcance entre dos puntos.

que el punto p_2 sea considerado como un outlier con respecto a C_2 sin hacer que el cluster C_1 también lo sea.

3.2.1. Definición formal de outliers locales

Como hemos visto, es necesaria otra forma de ver los outliers, una que nos permita evitar el problema de detectarlos cuando tenemos varios clusters de diferentes densidades. Para ello, lo primero que debemos hacer es introducir el concepto de distancia-$k(x_i)$, la cual definiremos como la distancia entre un punto $x_i \in X$ y otro $x_j \in X$ de tal manera que:

- Existen al menos k puntos $x_k \in X \setminus \{x_i\}$ que cumplen que $d(x_i, x_k) \leq d(x_i, x_j)$.
- Existen al menos $k-1$ puntos $x_k \in X \setminus \{x_i\}$ que cumplen que $d(x_i, x_k) < d(x_i, x_j)$.

A partir de esta definición podemos obtener el vecindario a distancia-k de un punto x_i, que está compuesto por todos aquellos puntos cuya distancia desde x_i es menor o igual a la distancia-k,

es decir, los k vecinos próximos de x_i. Eso lo podemos definir como, [4]

$$N_k(x_i) = \{x_j \in X \setminus \{x_i\} | d(x_i, x_j) \leq \text{distancia-}k(x_i)\}.$$

A continuación, debemos introducir dos nuevos conceptos, la distancia de alcance entre dos instancias o reach distance y la densidad de alcance local o local reachability distance. El primero de ellos, lo definiremos como el mayor valor entre la distancia-k y la distancia real, [4]

$$\text{r-dist}_k(x_i, x_j) = max\{\text{distancia-}k(x_j), d(x_i, x_j)\}. \tag{3.1}$$

Con esta formula conseguimos que todos los puntos de un mismo vecindario obtengan un valor de alcanzabilidad similar, evitando así posibles fluctuaciones estadísticas de $d(x_i, x_j)$ cuando x_i se encuentra cerca de x_j [4]. Un ejemplo sencillo del funcionamiento de esta distancia lo podemos ver en la figura 3.4, extraída de [4].

El otro concepto que se va a introducir es la densidad de alcance local, que es la inversa de la media de las distancias de alcance entre un punto y sus k vecinos más cercanos. Matemáticamente la podemos definir como [4]

$$lrd_k(x_i) = \frac{1}{\frac{1}{|N_k(x_i)|} \sum\limits_{x_j \in N_k(x_i)} \text{r-dist}_k(x_i, x_j)} = \frac{k}{\sum\limits_{x_j \in N_k(x_i)} \text{r-dist}_k(x_i, x_j)}. \tag{3.2}$$

Como estamos trabajando con los vecinos de un punto x_i, por la definición de la distancia de alcance (3.1), se cumple que $\text{r-dist}_k(x_i, x_j) = \text{distancia-}k(x_j)$, por lo que podemos reescribir la formula anterior como,

$$\begin{aligned} lrd_k(x_i) &= \frac{|N_k(x_i)|}{\sum\limits_{x_j \in N_k(x_i)} \text{r-dist}_k(x_i, x_j)} \\ &= \frac{k}{\sum\limits_{x_j \in N_k(x_i)} k\text{-distance}(x_j)} \end{aligned}$$

En esta ecuación es fácil ver que cuanto mayor sea la densidad de un cluster, mayor será el valor de lrd_k para un punto que se encuentre en su interior.

Finalmente, tras estas definiciones ya podemos dar una fórmula definitiva para asignar a cada punto un valor que nos indique su grado de outlier. Esta fórmula está definida como, [4]

$$LOF_k(x_i) = \frac{1}{|N_k(x_i)|} \sum_{x_j \in N_k(x_i)} \frac{lrd_k(x_j)}{lrd_k(x_i)}. \tag{3.3}$$

Al igual que antes, podemos utilizar las definiciones de distancia de alcance (3.1) y de densidad

de alcance local (3.2) para desarrollarla. Si hacemos esto nos queda la siguiente formula,

$$\begin{aligned} LOF_k(x_i) &= \frac{1}{|N_k(x_i)|} \sum_{x_j \in N_k(x_i)} \frac{lrd_k(x_j)}{lrd_k(x_i)} \\ &= \frac{1}{k \cdot lrd_k(x_i)} \sum_{x_j \in N_k(x_i)} lrd_k(x_j) \\ &= \frac{1}{k^2} \sum_{x_l \in N_k(x_i)} k\text{-distance}(x_l) \sum_{x_j \in N_k(x_i)} \frac{k}{\sum\limits_{x_m \in N_k(x_j)} k\text{-distance}(x_m)} \\ &= \frac{1}{k} \sum_{x_l \in N_k(x_i)} k\text{-distance}(x_l) \sum_{x_j \in N_k(x_i)} \frac{1}{\sum\limits_{x_m \in N_k(x_j)} k\text{-distance}(x_m)} \end{aligned}$$

De estas fórmulas, podemos extraer la siguiente conclusión. En el caso en que un punto se encuentre dentro de un cluster, su valor LOF será cercano a 1. Por otro lado, si los vecinos de un punto x_i se encuentran muy alejados de este o si los vecinos de los vecinos de un punto se encuentran muy cerca unos de otros, el valor LOF del punto x_i será bastante mayor que 1. De esta manera, cuanto mayor sea el valor de LOF, más anómalo consideraremos a un punto.

3.2.2. Límites superior e inferior al valor de LOF

Como ya hemos comentado, cuando un punto está dentro de un cluster C, es decir, un inlier, su valor de LOF va a ser cercano a 1. Esto lo podemos comprobar acotando el valor de LOF para los diferentes puntos de nuestro conjunto de datos. Primero definimos r-dist$_{min}$ y r-dist$_{max}$ como las distancias de alcance mínima y máxima de los puntos en C, tal que, [4]

$$\text{r-dist}_{min} = \text{mín}\{\text{r-dist}(x_i, x_j) | x_i, x_j \in C\},$$

$$\text{r-dist}_{max} = \text{máx}\{\text{r-dist}(x_i, x_j) | x_i, x_j \in C\}.$$

Para un punto x_i que se encuentra en C, que cumple que:

- todos los k vecinos próximos x_j de x_i se encuentran en C,
- todos los k vecinos próximos x_l de x_j también se encuentran en C

entonces el valor de LOF de x_i está acotado por [4]

$$\frac{\text{r-dist}_{min}}{\text{r-dist}_{max}} \leq LOF(x_i) \leq \frac{\text{r-dist}_{max}}{\text{r-dist}_{min}}.$$

Como podemos observar, si todos los puntos del cluster se encuentran cercanos entre sí, los valores r-dist$_{min}$ y r-dist$_{max}$ serán muy similares, por lo que el valor de LOF será 1, como queríamos comprobar.

Estos límites son válidos para puntos que se encuentren dentro de un cluster, pero también nos puede interesar acotar los valores de aquellos puntos que se encuentran fuera de ellos, es decir, los

posibles outliers. Para esto, primero debemos definir las distancias mínimas y máximas directas e indirectas de un punto x_i. La distancias directas son la menor y mayor r-dist entre x_i y sus vecinos mientras que las indirectas son la menor y mayor r-dist entre los vecinos x_j de x_i y los vecinos x_l de los diferentes x_j. Estas definiciones las podemos expresar como, [4]

$$\begin{aligned} direct_{min}(x_i) &= \text{mín}\{\text{r-dist}(x_i, x_j) | x_j \in N_k(x_i)\}, \\ direct_{max}(x_i) &= \text{máx}\{\text{r-dist}(x_i, x_j) | x_j \in N_k(x_i)\}, \\ indirect_{min}(x_i) &= \text{mín}\{\text{r-dist}(x_j, x_l) | x_j \in N_k(x_i) \text{ y } x_l \in N_k(x_j)\}, \\ indirect_{max}(x_i) &= \text{máx}\{\text{r-dist}(x_j, x_l) | x_j \in N_k(x_i) \text{ y } x_l \in N_k(x_j)\}. \end{aligned}$$

Con estas distancias, los nuevos límites para un punto x_i son [4]

$$\frac{direct_{min}(x_i)}{indirect_{max}(x_i)} \leq LOF(x_i) \leq \frac{direct_{max}(x_i)}{indirect_{min}(x_i)}. \tag{3.4}$$

Como ya hemos comentado, se puede dar la situación en la que un punto se encuentre entre varios clusters. En esta situación los límites anteriores no serían válidos, ya que necesitaríamos añadir cómo afecta cada conjunto de puntos al valor de LOF. Para hacer esto, si los $N_k(x_i)$ vecinos del punto x_i se encuentran repartidos en n clusters, debemos introducir la variable $\xi_i = |C_i|/|N_k(x_i)|$ que nos indica el porcentaje de vecinos de x_i que se encuentran en C_i. Con este nuevo concepto, los nuevos límites quedarían [4]

$$\left(\sum_{i=1}^{n} \xi_i \cdot direct^i_{min}(x_i)\right) \cdot \left(\sum_{i=1}^{n} \frac{\xi_i}{indirect^i_{max}(x_i)}\right) \leq LOF(x_i)$$

$$LOF(x_i) \leq \left(\sum_{i=1}^{n} \xi_i \cdot direct^i_{max}(x_i)\right) \cdot \left(\sum_{i=1}^{n} \frac{\xi_i}{indirect^i_{min}(x_i)}\right)$$

Como podemos observar, si en esta formula tenemos únicamente un cluster, queda igual que (3.4).

3.2.3. El impacto del parámetro k

Como podemos observar en (3.3), el único parámetro que podemos controlar en LOF es el número de vecinos a tener en cuenta a la hora de calcular los diferentes valores. Por tanto la elección de k es muy importante, ya que, si escogemos un valor muy pequeño podemos no estar teniendo en cuenta las características globales de la muestra y, por ejemplo, estar clasificando como normales un conjunto pequeño de outliers muy cercanos entre sí. Por otro lado, si escogemos un k demasiado grande podemos estar dejando de lado las características locales de los puntos, pudiendo darse el caso de tomar como normal un outlier en el que todos sus vecinos forman un cluster independiente.

Estas dos situaciones hacen que la elección de k no sea sencilla y dependa en gran medida del problema que estemos tratando. De esta manera en [4] se propone como solución escoger un rango para k y calcular el valor de LOF para cada uno de ellos y quedarnos con el de mayor valor, es decir,

$$LOF_{k^*}(x_i) = \text{máx}\{LOF_k(x_i) | k_{\text{LB}} \leq k \leq k_{\text{UB}}\},$$

donde, k_{LB} y k_{UB} son el el menor y el mayor de los posibles valores de k. De esta forma nos aseguramos de que si un punto ha sido considerado como outlier para un número de vecinos concretos siempre lo vamos a identificar como tal, y no va a quedar enmascarado por alguna de las situaciones comentadas anteriormente.

Esta forma de asignar el valor de LOF a los diferentes puntos, no está disponible en la librería `scikit-learn`. Aquí, únicamente podemos seleccionar un valor para k. Aún así, si queremos que el valor de LOF de los puntos venga dado por el máximo posible entre un rango de valores de k, lo podemos hacer con el siguiente método Python.

```
def LOF_range_k(X,k_min,k_max):

    lista_lof = np.zeros((X.shape[0],))

    for k in range(k_min,k_max+1):
        clf = LocalOutlierFactor(n_neighbors=k)
        y_pred = clf.fit_predict(X)
        lista_lof = np.min([lista_lof,clf.negative_outlier_factor_],axis=0)
    return lista_lof
```

Se utiliza el método `min` de la librería `NumPy` ya que los valores LOF en `scikit-learn` son negativos, por lo que el valor máximo es el más negativo.

3.3. Isolation forests

Recordamos que las anomalías o outliers son aquellos datos que presentan características diferentes a aquellos considerados como normales. La mayoría de modelos que tratan de identificarlas intentan definir qué es una instancia normal para después catalogar como anomalías a todas aquellas que no se ajustan a esta definición [5]. Este tipo de modelos presentan dos inconvenientes:

- Puesto que están optimizados para detectar datos normales, a la hora de detectar anomalías nos podemos encontrar con modelos poco eficaces.
- La complejidad de algunos de estos modelos aumenta con la dimension y el tamaño de los datos.

Para evitar esto, el algoritmo de Isolation Forests (iForests) se centra en aislar explícitamente las anomalías en lugar de definir los datos normales [14]. Para hacer esto se utiliza una estructura basada en árboles, ya que se puede construir de tal manera que todos los puntos se encuentren aislados. Como las anomalías tienen características diferenciadas, tenderán a quedarse más cerca de la raíz del árbol, mientras que las instancias normales tenderán a quedarse en zonas más profundas. A estos árboles se les llama Isolation Trees o iTrees.

El algoritmo de iForest utiliza un conjunto de iTrees para identificar como anomalías aquellos puntos que, de media, tienen poca profundidad. Algunas características de iForests son [14]:

- Como las anomalías tienden a quedarse en la parte superior del árbol, no es necesario construirlo completamente.
- Muestras pequeñas de datos producen buenos resultados, ya que reducen los efectos de empantanamiento o swamping (identificar erróneamente datos normales como anomalías)

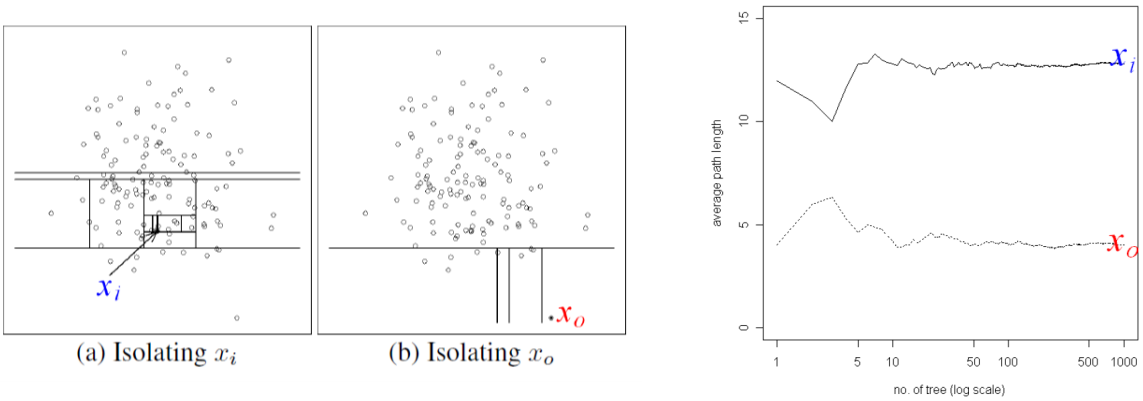

Figura 3.5: Aislamiento de dos puntos diferentes (izquierda) y profundidad media de dos puntos en función del número de árboles (derecha).

y masking o enmascaramiento (un grupo grande de anomalías puede ser identificado como normal).

- Al no utilizar medidas basadas en la distancia o en la densidad, tiene un bajo coste computacional.
- Presenta una complejidad temporal lineal con poco uso de memoria.
- Permite el uso de grandes conjuntos de datos con altas dimensiones.

3.3.1. Aislamiento e Isolation Trees

Para aislar las diferentes instancias, los iTrees van haciendo particiones de forma aleatoria hasta que todas las instancias se quedan aisladas. Este procedimiento hace que las anomalías queden más arriba en el árbol, ya que, al haber menos, se necesitan pocas particiones para aislarlas. Además, al tener características menos comunes son más propensas a quedar separadas en las particiones iniciales.

Debido a esto, cuando en un bosque de este tipo de árboles producen de media profundidades bajas para ciertos puntos, estos tienen una alta probabilidad de ser anomalías. En la figura 3.5, extraída de [14], podemos observar cómo para un punto normal se necesitan más particiones que para una anomalía.

Denominamos T a un nodo de un iTree, que puede ser un nodo externo sin hijos o un nodo interno con exactamente dos nodos hijos (T_l, T_r). Cada uno de los nodos internos está compuesto por un atributo q y un valor de división p tal que $q < p$ divide los puntos que se encuentran en el nodo en T_l y T_r.

Si disponemos de una muestra de datos $X = \{x_1, ..., x_n\}$ con n elementos pertenecientes a una distribución de dimensión d, entonces, utilizaremos una submuestra $X' \subset X$ con ψ puntos para construir un iTree. De forma recursiva iremos dividiendo X' seleccionando de forma aleatoria un atributo q y un valor p hasta que:

1. Únicamente quede una instancia en el nodo o,
2. Todos los datos del nodo tengan el mismo valor para el atributo q.

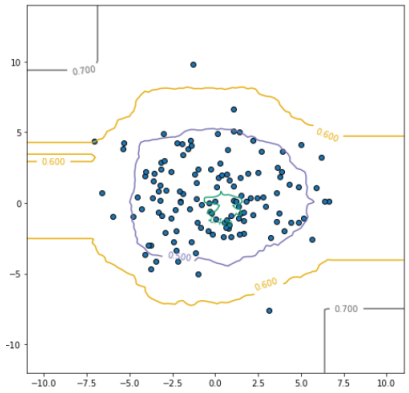

Figura 3.6: Ejemplo de curvas de nivel de s.

Definición (Profundidad)**.** *La profundidad $h(x)$ de un punto x es el número de ramas que hay que atravesar hasta llegar desde el nodo raíz hasta el nodo externo en el que se encuentra x.*

Una profundidad pequeña indica una alta susceptibilidad al aislamiento, mientras que una profundidad alta indica una baja susceptibilidad al aislamiento.

Anomaly score

Para decidir si un punto es una anomalía o no, necesitamos algún tipo de puntuación que nos ayude a tomar esta decisión. El problema de encontrar este tipo de medidas para un iTree en relación con el valor de $h(x)$ es que la altura máxima del iTree crece en el orden de ψ, mientras que la altura media crece en el orden de $\log(\psi)$. Si intentamos normalizar el valor de $h(x)$ en función de alguno de estos valores, nos encontraremos con que, o su valor no está limitado o que no es comparable directamente.

Para solucionar esto debemos fijarnos en que la estructura de un iTree es similar a la de un árbol binario de búsqueda, por lo que el valor medio de $h(x)$ para un nodo externo es la misma que una búsqueda fallida. Para un conjunto de datos ψ, este valor es

$$c(\psi) = \begin{cases} 2H(\psi - 1) - (2(\psi - 1)/n) & \text{para } \psi > 2, \\ 1 & \text{para } \psi = 2, \\ 0 & \text{en cualquier otro caso,} \end{cases} \tag{3.5}$$

donde $H(i)$ es el i-ésimo número armónico, que puede ser estimado como $\ln(i) + 0{,}5772156649$ (constante de Euler).

Como $c(\psi)$ es la media de $h(x)$ dado ψ, podemos utilizarlo para normalizar $h(x)$, obteniendo así una puntuación o anomaly score $s(x, \psi)$ para un punto x,

$$s(x, \psi) = 2^{-\dfrac{E(h(x))}{c(\psi)}},$$

donde $E(h(x))$ es la media de $h(x)$ en un conjunto de iTrees.

En función del valor de s, podemos podemos hacer las siguientes afirmaciones:

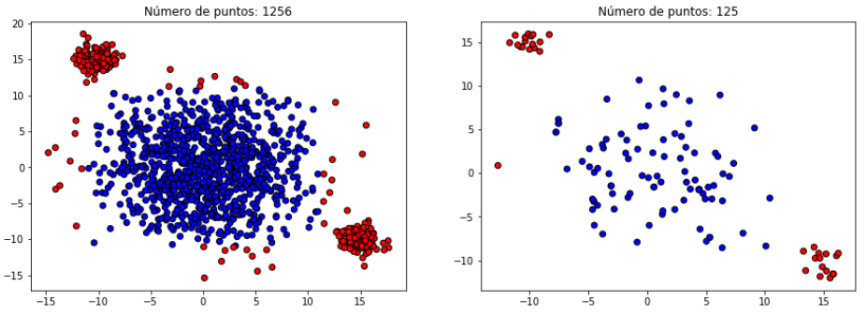

Figura 3.7: Ejemplo de submuestreo.

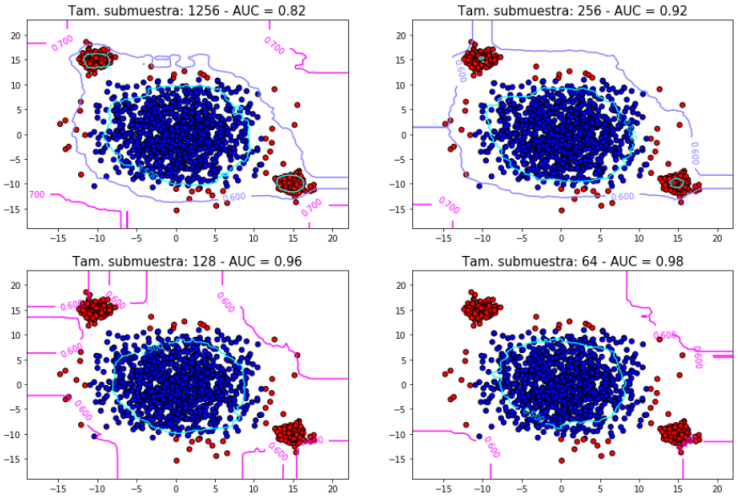

Figura 3.8: Curvas de nivel de s para diferentes tamaños de muestra.

- Si el valor de s es cercano a 1, entonces estamos frente a una anomalía.
- Si el valor de s es menor de 0.5, podemos decir con seguridad que se trata de un dato normal.
- Si todos los puntos tienen un valor de $s \approx 0{,}5$, el conjunto de datos no presenta grandes anomalías. Esto se debe a que todos los puntos se encuentran aproximadamente a la misma altura, por lo que $E(h(x)) \sim c(\psi)$.

En la figura 3.6 podemos ver un ejemplo del comportamiento de este valor en un conjunto pequeño de datos. Los puntos que se encuentran dentro de cada una de las líneas tienen un valor de s menor o igual al indicado sobre la línea. Podemos afirmar que los puntos con $s > 0{,}6$ son potenciales anomalías.

Swamping y enmascaramiento

Dentro del campo de detección de anomalías, dos problemas que se presentan de manera frecuente son los de empantanamiento o swamping y el de masking o enmascaramiento. El primero se refiere al hecho de identificar como anomalía un dato normal. Suele suceder cuando hay muchos datos normales o cuando los que hay se encuentran dispersos. El segundo problema consiste en la

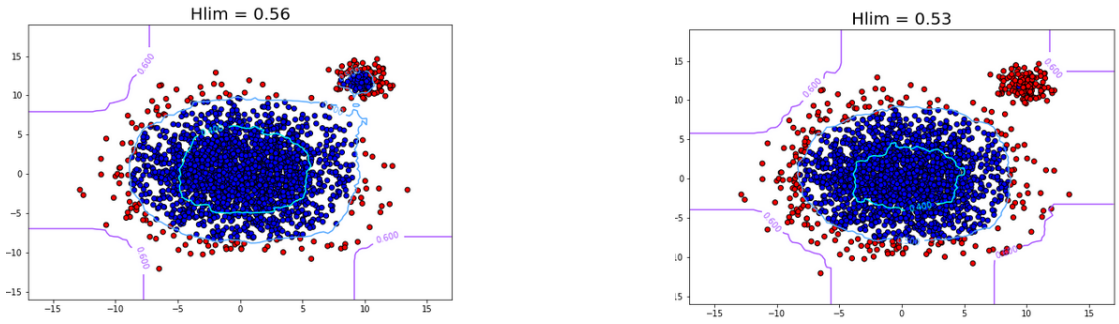

Figura 3.9: Ejemplo de la modificación de la altura máxima.

existencia de tantas anomalías que se camuflan como instancias normales. Esto pasa cuando los clusters de anomalías se vuelven grandes y densos.

Para solventar estos problemas, iForest permite construir un modelo utilizando diferentes submuestras, para que así, a cada árbol le resulte más sencillo aislar las diferentes anomalías. Esto lo podemos ver en la figura 3.7, en la que podemos observar que al tomar una submuestra del 10 % de los datos aunque mantenemos la estructura original, los clusters son menos densos.

La mejoría que proporciona este procedimiento lo podemos ver en la figura 3.8, donde se han construido varios modelos modificando el tamaño de la submuestra que utilizan los árboles. Al evaluar los modelos mediante la métrica de área bajo la curva (AUC por sus siglas en inglés) podemos comprobar que reduciendo el tamaño de la submuestra el valor AUC aumenta.

Ajuste de la granularidad del valor de la anomaly score

Para decidir si un cluster de datos es normal o si se trata de un conjunto de anomalías, iForest es capaz de modificar la altura máxima hasta la que puede llegar un árbol al calcular la profundidad a la que se encuentra un punto. De esta manera, al reducir el límite máximo de altura estaremos impidiendo que el cluster de anomalías genere mucha profundidad. A esto se lo conoce como disminuir la granularidad de los valores de la anomaly score. En cambio, al aumentar el límite máximo, podemos detectar puntos aislados que rodean a los clusters.

Dependiendo del paquete que estemos utilizando para ejecutar este algoritmo, nos podemos encontrar con que no tenemos la posibilidad de modificar este parámetro directamente, como pasa en `scikit-learn`. Aquí, el parámetro que más se asemeja es el `offset_` que se suma al inverso de la anomaly score de tal manera que las instancias normales tengan un valor positivo y las anomalías uno negativo. Este parámetro lo podemos utilizar como umbral a la hora de decidir a partir de qué valor consideramos un punto como anomalía. Aumentar este valor hace que un punto tenga que tener una anomaly socre muy alta para ser considerado como anomalía, es decir, tiene que tener muy poca profundidad. En la figura 3.9 podemos ver cómo afecta este parámetro en la detección de anomalías. Como podemos observar, al aumentar el valor del `offset_` estamos detectando como normales puntos que se encuentran dentro del cluster anómalo.

3.3.2. Detección de anomalías con iForest

El proceso de detección de anomalías con iForest tiene dos etapas:

Algorithm 1 : $iForest(X, t, \psi)$

Inputs: X - input data, t - number of trees, ψ - subsampling size
Output: a set of t *iTrees*

```
Initialize Forest
for i = 1 to t do
    X' ← sample(X, ψ)
    Forest ← Forest ∪ iTree(X')
end for
return Forest
```

Algorithm 2 : $iTree(X')$

Inputs: X' - input data
Output: an iTree

```
if X' cannot be divided then
    return exNode{Size ← |X'|}
else
    let Q be a list of attributes in X'
    randomly select an attribute q ∈ Q
    randomly select a split point p between the max and min values of attribute
    q in X'
    X_l ← filter(X', q < p)
    X_r ← filter(X', q ≥ p)
    return inNode{Left ← iTree(X_l),
                  Right ← iTree(X_r),
                  SplitAtt ← q,
                  SplitValue ← p}
end if
```

Figura 3.10: Algoritmos de entrenamiento para iForest e iTree.

- Entrenamiento, donde construimos diferentes iTrees mediante submuestras del conjunto de entrenamiento.
- Evaluación, donde utilizamos el conjunto de evaluación para obtener una anomaly socre para cada punto.

Etapa de entrenamiento

Los diferentes iTrees se construyen dividiendo el conjunto de instancias dado hasta que cada una de ellas se encuentra aislada. Para ello, utilizamos una submuestra $X' \subset X$ extraída de forma aleatoria sin remplazamiento de X. En esta etapa construimos un conjunto de t árboles a partir de ψ datos. Este proceso tiene una complejidad de $O(t\psi \log \psi)$ [14].

Los algoritmos propuestos en [14] para el entrenamiento los podemos ver en la figura 3.10.

Etapa de evaluación

En esta etapa, para cada punto x se calcula su profundidad $h(x)$ contando el número de ramas e que hay que pasar desde la raíz hasta la hoja en la que se encuentra y sumando un valor $c(size)$ calculado a partir de (3.5), siendo $size$ el tamaño del iTree. Este cálculo lo hacemos en cada uno de los iTrees del iForest y finalmente obtenemos la anomaly score del punto en cuestión.

Algorithm 3 : $PathLength(x, T, hlim, e)$

Inputs : x - an instance, T - an iTree, $hlim$ - height limit, e - current path length; to be initialized to zero when first called

Output: path length of x

1: **if** T is an external node or $e \geq hlim$ **then**
2: return $e + c(T.size)$ {$c(.)$ is defined in Equation 1}
3: **end if**
4: $a \leftarrow T.splitAtt$
5: **if** $x_a < T.splitValue$ **then**
6: return $PathLength(x, T.left, hlim, e + 1)$
7: **else** $\{x_a \geq T.splitValue\}$
8: return $PathLength(x, T.right, hlim, e + 1)$
9: **end if**

Figura 3.11: Algoritmo de evaluación para iForest.

El algoritmo para esta etapa propuesto en [14] lo podemos ver en la figura 3.11.

3.4. One-class SVM

Durante los últimos años, han surgido numerosas técnicas de aprendizaje supervisado basadas en la utilización de kernels. Estos desarrollos se han dado sobre todo en los campos de reconocimiento de patrones y problemas de regresión [6]. Esta popularidad ha hecho que se intente utilizar este tipo de algoritmos en el aprendizaje no supervisado, que se puede caracterizar por la búsqueda y estimación de funciones sobre los datos, con el fin de obtener cómo se distribuyen. Este tipo de aprendizaje puede ser visto como la búsqueda de la función de densidad de los datos, ya que conociendo esta información, podemos resolver, esencialmente, cualquier problema sobre los mismos. El método de one-class SVM pretende hacer justamente eso, encontrar una función binaria que delimite la región en la que la mayoría de los datos de entrada se encuentran.

3.4.1. Antecedentes

Si $x_1, x_2, ..., x_n$ son variables aleatorias independientes e idénticamente distribuidas en un conjunto X con una distribución P, $\mathcal{C}$ es una clase de subconjuntos medibles de X y λ es una función real definida en $\mathcal{C}$, entonces la función de los cuantiles con respecto a $(P, \lambda, \mathcal{C})$ es [6]

$$U(\alpha) = \inf\{\lambda(C) : P(C) \geq \alpha, C \in \mathcal{C}\} \quad 0 < \alpha \leq 1.$$

Si denominamos $C(\alpha)$ como el conjunto $C \in \mathcal{C}$ que contiene el ínfimo y seleccionamos λ como la medida de Lebesgue, entonces $C(\alpha)$ es el volumen mínimo $C \in \mathcal{C}$ que contiene al menos una fracción α de la masa de probabilidad [6].

El objetivo de one-class SVM es encontrar regiones cercanas a $C(\alpha)$. Para ello se minimiza un modelo basado en vectores de soporte con el que, mediante el uso de kernels, podemos controlar la función que describe C. En un espacio multidimensional, podemos ver este proceso como la utilización de $\lambda(C_w) = ||w||^2$, donde $C_w = \{x : f_w(x) \geq \rho\}$ y (w, ρ) son el vector de pesos y un offset que definen un hiperplano en el espacio asociado al kernel [6].

3.4.2. Formulación de la One-class SVM

Este modelo está definido por una función f que otorga el valor de $+1$ a la región en la que se encuentran la mayoría de los puntos y -1 en cualquier otra parte. Para hacer esto, la estrategia principal es transformar el espacio de entrada $\mathcal{R}$ de los datos a un espacio de mayor dimensión F mediante una función $\Phi(x_i)$ para poder separar los datos mediante el hiperplano de mayor margen. Estas transformaciones podremos realizarlas mediante la utilización de un kernel, como puede ser el gaussiano,

$$k(x, y) = e^{-||x-y||^2/c}.$$

Para encontrar hiperplano que separa los datos debemos resolver el siguiente problema cuadrático [6]

$$\min_{w \in F, \xi \in \mathbb{R}^n, \rho \in \mathbb{R}} \frac{1}{2}||w||^2 + \frac{1}{\nu n}\sum_i \xi_i - \rho, \tag{3.6}$$

sujeto a $w \cdot \Phi(x_i) \geq \rho - \xi_i,\ \xi_i \geq 0$ y $\nu \in (0, 1]$. Puesto que las variables de holgura o slack ξ_i son penalizadas en la función objetivo, podemos esperar que si w y ρ resuelven el problema, entonces, la función de decisión

$$f(x) = \text{sign}((w \cdot \Phi(x)) - \rho)$$

será positiva para la mayoría de x_i del conjunto de entrenamiento, mientras que el término $||w||^2$ se mantendrá pequeño [6]. Estas dos condiciones las podemos controlar mediante la variable ν.

Si utilizamos unos multiplicadores $\alpha_i, \beta_i \geq 0$, podemos obtener el Lagrangiano de la función objetivo como

$$L(w, \xi, \rho, \alpha, \beta) = \frac{1}{2}||w||^2 + \frac{1}{\nu n}\sum_i \xi_i - \rho - \sum_i \alpha_i((w \cdot \Phi(x_i)) - \rho + \xi_i) - \sum_i \beta_i \xi_i.$$

Si calculamos las derivadas parciales del Lagrangiano con respecto a las variables w, ξ y ρ, y las igualamos a 0, obtenemos que

$$w = \sum_i \alpha_i \Phi(x_i),$$

$$\alpha_i = \frac{1}{\nu n} - \beta_i \leq \frac{1}{\nu n},$$

$$\sum_i \alpha_i = 1.$$

Si sustituimos estos valores en el Lagrangiano, obtenemos el problema dual

$$\min \frac{1}{2}\sum_{ij} \alpha_i \alpha_j k(x_i, x_j) \tag{3.7}$$

sujeto a $0 \leq \alpha_i \leq \frac{1}{\nu n}$ y $\sum_i \alpha_i = 1$. Además, en el caso óptimo se han de cumplir las condiciones KKT, que son las siguientes.

$$(w^* \cdot \Phi(x_i)) - \rho^* + \xi_i^* \geq 0,$$

$$\alpha_i^* \geq 0,$$

$$\alpha_i^*[(w^* \cdot \Phi(x_i)) - \rho^* + \xi_i^*] = 0,$$
$$\xi_i^* \geq 0,$$
$$\beta_i^* \geq 0,$$
$$\beta_i^* \xi_i^* = 0.$$

Cuando α_i y β_i son mayores que 0, podemos calcular ρ como

$$\rho^* = (w^* \cdot \Phi(x_i)) = \sum_j \alpha_j^* k(x_j, x_i).$$

Después de esto, podemos reescribir la función de decisión como

$$f(x) = \text{sign}\left(\sum_i \alpha_i k(x_i, x) - \rho\right). \tag{3.8}$$

En este algoritmo, si ν tiende a 0, el límite superior de la desigualdad de la restricción del problema dual desaparece. Esto hace que el problema se asemeje al de márgenes duros, ya que la penalización de los errores es infinita, como se puede ver en el problema primal (3.6).

Otro punto de vista desde el que ver este problema es mediante la utilización de esferas mínimas que contengan los datos [6]. En este caso el problema a solucionar es

$$\min_{R\in\mathbb{R},\xi\in\mathbb{R}^n,c\in F} R^2 + \frac{1}{\nu n}\sum_i \xi_i,$$

sujeto a $||\phi(x_i) - c||^2 \leq R^2 + \xi_i$ y $\xi_i \geq 0$ para todo $i \in [n]$. El dual de este problema es

$$\min_\alpha \sum_{ij} \alpha_i \alpha_j k(x_i, x_j) - \sum_i \alpha_i k(x_i, x_i), \tag{3.9}$$

que, en el caso óptimo, está sujeto a las condiciones KKT $\forall i$,

$$R^{*\,2} + \xi_i^* - ||\phi(x_i) - c^*||^2 \geq 0,$$
$$\alpha_i^* \geq 0,$$
$$\alpha_i^*[R^{*\,2} + \xi_i^* - ||\phi(x_i) - c^*||^2] = 0,$$
$$\xi_i^* \geq 0,$$
$$\beta_i^* \geq 0,$$
$$\beta_i^* \xi_i^* = 0.$$

De aquí, para un punto x_i cualquiera, cuando $\alpha_i^* > 0$ y $\beta_i^* > 0$, podemos obtener el valor de R^* como

$$R^* = \sqrt{k(x_i, x_i) - 2\sum_p \alpha_p^* k(x_p, x_i) + \sum_{pq} \alpha_p^* \alpha_q^* k(x_p, x_q)}.$$

Finalmente, la función de decisión de este problema es

$$f(x) = \text{sign}\left(R^2 - \sum_{ij} \alpha_i \alpha_j k(x_i, x_j) + 2\sum_i \alpha_i k(x_i, x) - k(x, x)\right). \tag{3.10}$$

Si utilizamos un kernel $k(x, y)$ que dependa únicamente de $x - y$, entonces, $k(x, x) = cte$. En este caso, la restricción de igualdad implica que el término lineal de la función dual es constante, por lo que los duales ((3.7) y (3.9)) de ambos puntos de vista son equivalentes,

$$\frac{1}{2}\sum_{ij} \alpha_i \alpha_j k(x_i, x_j) \sim \sum_{ij} \alpha_i \alpha_j k(x_i, x_j) - cte.$$

De esta manera, si los problemas duales son equivalentes, las funciones de decisión ((3.8) y (3.10)) también lo serán.

3.4.3. Algoritmo de optimización

Como ya se ha comentado anteriormente, la forma de encontrar los valores óptimos de las diferentes α en SVM es mediante el algoritmo de SMO. En el caso de One-class SVM también podemos aplicar dicho algoritmo a partir de la fórmula (3.7), que está sujeta a las restricciones $0 \leq \alpha_i \leq \frac{1}{\nu n}$ y $\sum_i \alpha_i = 1$.

Debido a la restricción de suma, en este algoritmo debemos ir actualizando los coeficientes, como mínimo, de dos en dos. En este caso, vamos a escoger dos coeficientes α_L^t y α_U^t. Los valores de estos coeficientes se van a ir modificando mediante las fórmulas

$$\alpha_{L^t}^{t+1} = \alpha_{L^t}^t + \delta_{L^t}^t, \quad \alpha_{U^t}^{t+1} = \alpha_{U^t}^t + \delta_{U^t}^t,$$

mientras que el resto de α_j se mantendrán constantes. Por comodidad a la hora de realizar los cálculos, la restricción de igualdad nos permite expresar el incremento de α_{L^t} en función del de α_{U^t} como

$$\delta_{L^t}^t = -\delta_{U^t}^t.$$

Para facilitar los cálculos, vamos a hacerlos en notación vectorial, por lo que podemos expresar el problema dual como

$$\Theta(\alpha) = \frac{1}{2}\alpha^t K \alpha,$$

donde α es el vector con los diferentes valores y K es la matriz cuyos valores $K_{ij} = k(x_i, x_j)$. De manera análoga podemos expresar la modificación de los valores de los coeficientes como

$$\alpha^{t+1} = \alpha^t + \delta(e_L - e_U) = \alpha^t + \delta D_{LU},$$

donde e_L y e_U son vectores formados por 0 excepto en las posiciones L y U, donde hay un 1.

Una vez hecho esto, para un instante $t+1$ sustituimos los valores de los coeficientes en la fórmula

del problema dual, obtenemos que

$$\begin{aligned}\Theta(\alpha^{t+1}) &= \frac{1}{2}(\alpha + \delta D_{LU})^T K(\alpha + \delta D_{LU}) \\ &= \frac{1}{2}\alpha^T K\alpha + \frac{1}{2}\delta^2 D_{LU}^T K D_{LU} + \delta\alpha^T K D_{LU}.\end{aligned}$$

Si comparamos esta solución con la de un instante anterior $\Theta(\alpha^t)$ podemos observar que la ganancia entre ambas es de

$$\Psi(\delta) = \frac{1}{2}\delta^2 D_{LU}^T K D_{LU} + \delta\alpha^T K D_{LU}.$$

Puesto que lo que nos interesa es encontrar la solución en el menor número de iteraciones posibles, debemos maximizar esta ganancia, por lo que debemos calcular su derivada e igualarla a 0, lo que nos da un valor óptimo de de δ,

$$\delta^* = -\frac{\alpha^T K D_{LU}}{D_{LU}^T K D_{LU}}.$$

Al sustituir este valor en la fórmula de la ganancia obtenemos que

$$\begin{aligned}\Psi(\delta^*) &= \frac{1}{2}\frac{(\alpha^T K D_{LU})^2}{(D_{LU}^T K D_{LU})^2} D_{LU}^T K D_{LU} - \frac{\alpha^T K D_{LU}}{D_{LU}^T K D_{LU}}\alpha^T K D_{LU} \\ &= -\frac{1}{2}\frac{(\alpha^T K D_{LU})^2}{D_{LU}^T K D_{LU}}.\end{aligned}$$

Como ya hemos comentado, nuestro objetivo es maximizar la ganancia, por lo que los índices L y U deben maximizar $(\alpha^T K D_{LU})^2$. Para que esto suceda, los criterios de selección de ambos índices serán los siguientes,

$$\begin{aligned}L &= \underset{\alpha_i < \frac{1}{\nu n}}{\operatorname{argmin}}\{\alpha^T K e_i\}, \\ U &= \underset{\alpha_j > 0}{\operatorname{argmax}}\{\alpha^T K e_j\}.\end{aligned}$$

Las restricciones en la elección de los índices L y U se deben a que los valores de las α asociadas a ellos van a crecer en el caso del primero mientras que van a descender en el caso del segundo y, como ya hemos visto, se debe cumplir la condición $0 \leq \alpha_i \leq \frac{1}{\nu n}$.

El valor óptimo obtenido de esta manera para δ^* es válido en el caso de que no se viole ninguna de las restricciones, pero se puede dar el caso en el que este valor haga que alguno de los coeficientes no cumpla con la condición de caja. Es decir, que sea mayor que $\frac{1}{\nu n}$ o menor que 0. Para solucionar esto, el valor elegido de $\overline{\delta}$ debe ser

$$\overline{\delta} = \text{mín}\{\delta^*, \frac{1}{\nu n} - \alpha_L, \alpha_U\}.$$

Haciendo esto nos aseguramos de que las restricciones siempre se van a cumplir y por tanto estaremos obteniendo los coeficientes óptimos.

3.4.4. Resultados teóricos

A continuación, vamos a ver el algoritmo de one-class SVM desde un punto de vista teórico.

Definición. *Denominamos como separable a una muestra $x_1, ..., x_n$ si existe algún $w \in F$ que cumpla $w \cdot x_i > 0$ para todo $i \in [n]$ [6].*

Proposición. *Si un conjunto de datos es separable, entonces existe un único hiperplano con las siguientes propiedades [6]:*

- *Separa todos los datos del origen.*
- *Su distancia al origen es máxima con respecto a todos los hiperplanos.*

Para un $\rho > 0$, el hiperplano se obtiene como

$$\min_{w \in F} \frac{1}{2}||w||^2 \ \ \textit{sujeto a } w \cdot x_i \geq \rho, \ \ i \in [n].$$

Esta proposición la podemos demostrar de la siguiente manera. Si $\pi = w \cdot x + b$ es un hiperplano y w es ortogonal a él, podemos definir la distancia entre cualquier punto y el hiperplano como

$$d(x, \pi) = \frac{|w \cdot x + b|}{||w||}.$$

Podemos definir el margen como la distancia mínima entre los datos y el hiperplano π, que lo podemos expresar como

$$\min_i d(x_i, \pi) = \min_i \frac{y_i(w \cdot x + b)}{||w||} = \frac{\rho}{||w||}.$$

Como lo que estamos buscando es maximizar el margen, tenemos que resolver la siguiente ecuación,

$$\max \frac{\rho}{||w||} \equiv \min ||w|| \equiv \min \frac{1}{2}||w||^2 \quad \text{s.t. } y_i(w \cdot x + b) \geq \rho,$$

que es la misma que se da en la proposición.

En el caso de problemas no separables, además de los puntos que están en el otro lado del hiperplano, también consideraremos como outliers los puntos que se encuentran dentro del margen del hiperplano.

Proposición. *Si asumimos que las soluciones al problema primal y dual cumplen que $\rho \neq 0$, entonces podemos afirmar que [6]:*

- *ν es un límite superior para la fracción de outliers.*
- *ν es un límite inferior para la fracción de vectores de soporte.*

4

Resultados experimentales

A lo largo de este capítulo vamos a comprobar el funcionamiento de los diferentes métodos de detección de outliers y anomalías. Esto lo vamos a hacer desde dos puntos de vista, mediante la utilización de conjuntos de datos sintéticos y mediante la utilización de datos reales. Para ello tenemos disponibles diferentes librerías, como PyOD propuesta en [12] o `scikit-learn`. De estas dos, vamos a utilizar la segunda.

Además de esto, en este capítulo se propone una forma de mejorar la hiperparametrización de los modelos de detección de outliers. Esta forma consiste en la creación de un estimador que combine un modelo de detección de outliers y otro de predicción, ya sea para problemas de clasificación o de regresión.

Para abordar todo esto, primero vamos a ver una descripción de los modelos de detección de anomalías disponibles en `scikit-learn` en la sección 4.1. Después veremos el funcionamiento básico de cada uno de ellos sobre dos problemas sintéticos en la sección 4.2. A continuación, procederemos a la explicación detallada del método de hiperparametrización propuesto en la sección 4.3 y de la metodología a utilizar en los siguientes experimentos. Finalmente se exponen los resultados obtenidos en los experimentos realizados en problemas reales. Veremos un problema de clasificación en la sección 4.4 y dos de regresión en las secciones 4.5 y 4.6.

4.1. Modelos de scikit-learn para detección de outliers

En esta sección, se van a mostrar las implementaciones de los diferentes métodos de detección de anomalías vistos en el capítulo 3. Para ello se va a utilizar la librería de Python `scikit-learn`. Dentro de este paquete, se van a utilizar los estimadores `EllipticEnvelope`, `LocalOutlierFactor`, `IsolationForest` y `OneClassSVM`. A continuación se describen los principales parámetros de cada uno de ellos:

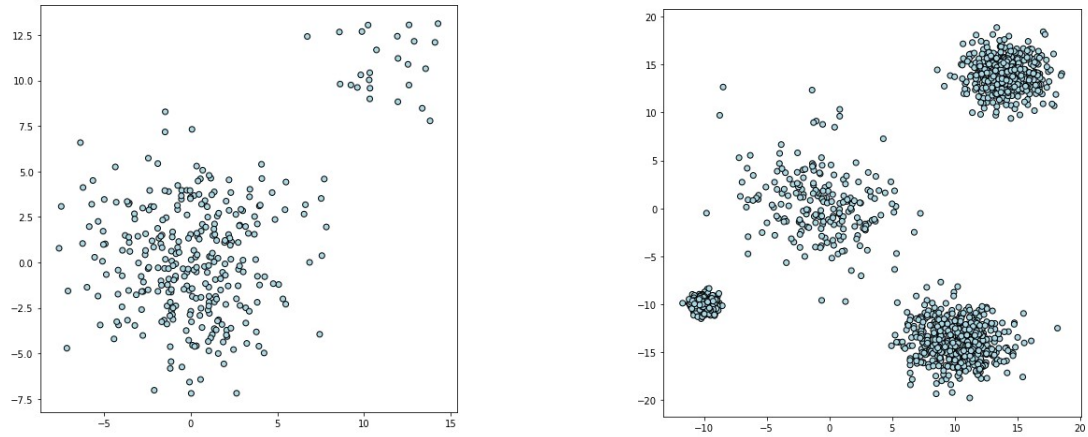

Figura 4.1: Problema sintético 1 (izquierda) y Problema sintético 2 (derecha).

4.1.1. EllipticEnvelope

Esta clase es la implementación de `scikit-learn` del método del mínimo determinante de la covarianza o Minimum Covariance Determinant (MCD), descrito en la sección 3.1. Los principales parámetros a utilizar son:

- `support_fraction`: es el equivalente al parámetro h descrito en la sección 3.1. Se trata de un valor en el rango $[0, 1]$ que indica la proporción de puntos a incluir en el elipsoide a minimizar. El valor por defecto es la proporción equivalente al valor propuesto en [3], que es
$$h = \frac{[patrones + features + 1]}{2}.$$
- `contamination`: indica la cantidad de contaminación, es decir, la proporción de outliers que contiene el conjunto de los datos. Por defecto, su valor es 0.1, lo que quiere decir que se detectaran como anomalías aproximadamente un 10 % de los datos.

4.1.2. LocalOutlierFactor

Esta clase es la implementación de `scikit-learn` del método Local outlier factor, descrito en la sección 3.2. Los principales parámetros a utilizar son:

- `n_neighbors`: es el número de vecinos a tener en cuenta por el algoritmo.
- `contamination`: al igual que en el modelo anterior, nos indica la cantidad de contaminación que hay en la muestra.

4.1.3. IsolationForest

Esta clase es la implementación de `scikit-learn` del método de Isolation Forest, descrito en la sección 3.3. Los principales parámetros a utilizar son:

- `n_estimators`: es el número de árboles a construir.
- `max_samples`: indica el número máximo de patrones utilizados en cada uno de los árboles. Por defecto se utilizan 256 patrones.
- `max_features`: indica el número de variables utilizadas en cada uno de los árboles. Por defecto se utilizan todas las disponibles.
- `contamination`: como en los métodos anteriores representa la proporción de outliers que nos vamos a encontrar.

En este modelo disponible en `scikit-learn`, hay que destacar la ausencia de un parámetro que nos permita controlar la altura máxima de los diferentes árboles. Puesto que este modelo se basa en el modelo de Random Forest, también echamos en falta algunos parámetros que nos permitan controlar el número de puntos necesarios para dividir un nodo o el número mínimo de muestras necesarias para que un nodo sea considerado una hoja.

4.1.4. OneClassSVM

Esta clase es la implementación de `scikit-learn` de One Class SVM, descrito en la sección 3.4. Los principales parámetros a utilizar son:

- `kernel`: es el núcleo a utilizar para el cálculo del hiperplano. Por defecto se utiliza el núcleo gaussiano.
- `gamma`: es el coeficiente γ del núcleo gaussiano. Por defecto su valor es de $1/d$, donde d es el número de variables.
- `nu`: representa la cota superior para los errores de entrenamiento y la cota inferior para el número de vectores de soporte. Su valor por defecto es de 0.5. Este parámetro está relacionado con la variable C de las máquinas de vectores de soporte originales mediante la fórmula $\nu = \frac{1}{nC}$, donde n es el número de puntos con el que estamos entrenando el modelo.

4.2. Experimentos no supervisados

En este apartado se van a estudiar dos problemas sintéticos con los diferentes modelos. Estos problemas, al tener únicamente dos variables, son sencillos de representar de forma gráfica, por lo que nos permiten visualizar mejor el efecto de los diferentes parámetros.

El primero de ellos, que lo podemos ver en la figura 4.1 a la izquierda, consta de un cluster grande (300 puntos) y otro más pequeño (25 puntos). En este caso, el objetivo es que los modelos detecten como outliers los puntos más externos del cluster de mayor tamaño y el cluster de menor tamaño en su totalidad.

El segundo problema (figura 4.1 a la derecha) aumenta la dificultad de detección ya que consta de 4 clusters de diferentes densidades y tamaños. En este caso, la solución que buscamos es aquella que detecte como outliers los puntos que no se pueden asociar a ningún cluster en concreto y los más externos de los clusters con mayor dispersión.

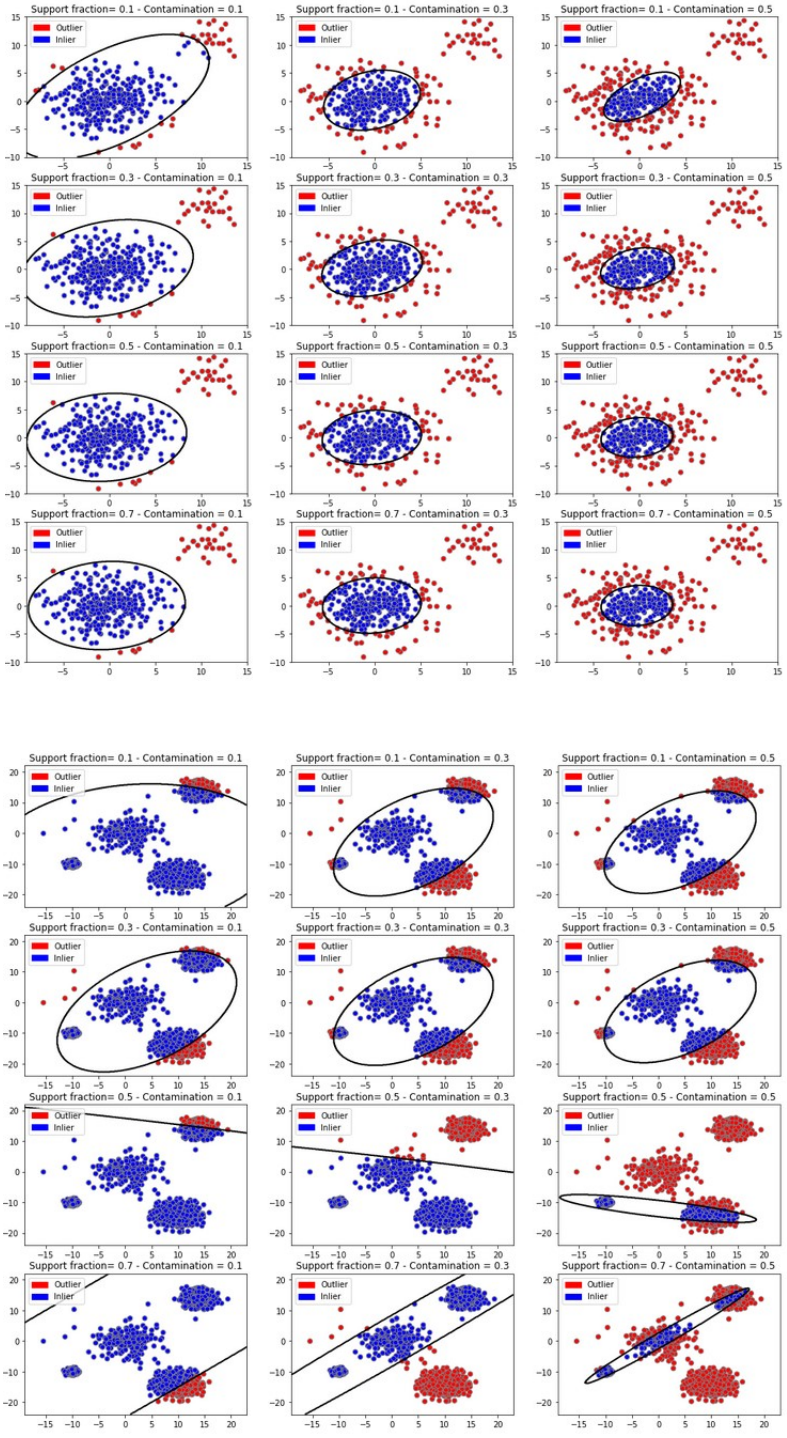

Figura 4.2: Resultados al problema sintético 1 (arriba) y al problema sintético 2 (abajo) mediante MCD.

4.2.1. MCD

Para este modelo, vamos a ver diferentes resultados en función de los parámetros `contamination` y `support_fraction`. Por comodidad, los resultados se van a exponer en forma de matriz con los

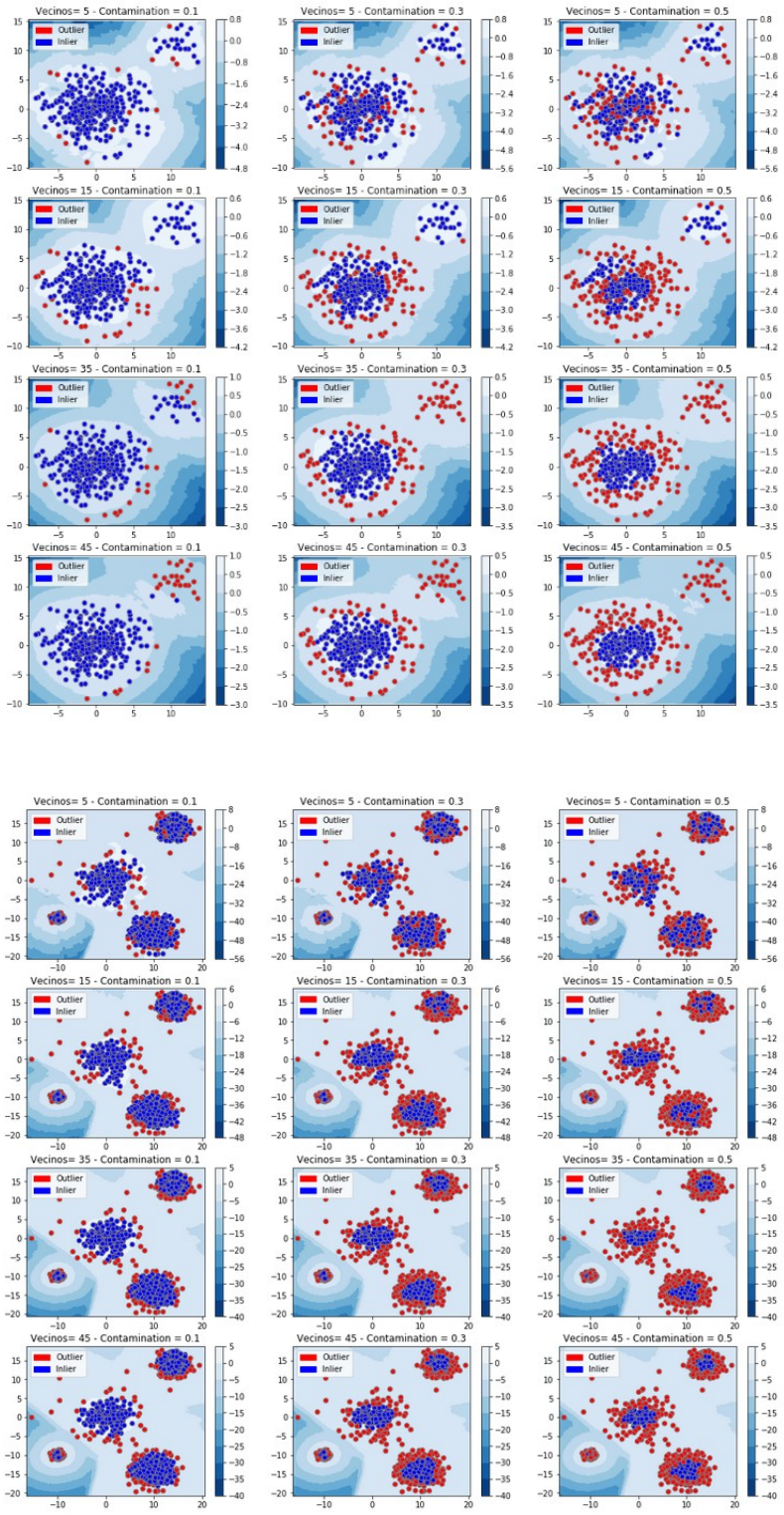

Figura 4.3: Resultados al problema sintético 1 (arriba) y al problema sintético 2 (abajo) mediante LOF.

valores de `contamination` (0.1, 0.3 y 0.5) creciendo en el eje x de izquierda a derecha y los valores de `support_fraction` (0.1, 0.3, 0.5 y 0.7) creciendo en el eje y de arriba a abajo.

En el caso del primer problema, como podemos observar en la figura 4.2 (arriba), los resultados que más se acercan al objetivo buscado para este problema los obtenemos con los valores de 0.1 para `contamination` y 0.3 para `support_fraction`. Si nos fijamos en el resto de combinaciones, cuando aumentamos la contaminación, estamos forzando a que se detecten demasiados outliers, haciendo que la elipse sea más pequeña. Por otro lado, al aumentar el valor de `support_fraction`, no observamos grandes diferencias en la solución obtenida. Esto se debe a que al finalizar el algoritmo de MCD, los resultados obtenidos son ponderados (ver apartado 3.1.3), lo que hace que aun con parámetros diferentes obtengamos soluciones muy similares.

En el caso del segundo problema, los resultados obtenidos (figura 4.2, abajo) no son los esperados. Esto se debe a que al haber cuatro clusters, no es posible encontrar una elipse que incluya los diferentes inliers de los clusters dejando fuera los outliers. Quizá, los casos más aproximados al resultado esperado serían los que tienen como valores `contamination = 0.3` y `support_fraction = 0.7` o `contamination = 0.5` y `support_fraction = 0.7`, ya que intentan capturar los inliers de tres de los cuatro clusters. Aún así, tampoco serían válidos puesto que se están considerando como inliers puntos que se encuentran a medio camino entre clusters y se está marcando como outlier un cluster completo.

4.2.2. LOF

En este caso, los parámetros a estudiar son `contamination` y `n_neighbors`. Al igual que en el caso anterior, los resultados se van a exponer en forma de matriz con los valores de `contamination` (0.1, 0.3 y 0.5) creciendo en el eje x de izquierda a derecha y los valores de `n_neighbors` (5, 15, 35 y 45) creciendo en el eje y, de arriba a abajo.

De las diferentes soluciones que podemos ver en la figura 4.3 (arriba) para el primer problema, la que nos ofrece el resultado más cercano al que esperábamos es la que tiene como valores `contamination = 0.1` y `n_neighbors = 45`, aunque como podemos ver, se están detectando como inliers dos puntos del cluster de outliers. Afinando un poco la búsqueda llegaríamos a un resultado en el que estos puntos sí que serían detectados como outliers. Aquí, la variación de la contaminación vuelve a hacer que nos encontremos con más outliers de los esperados, a veces incluso dentro de los clusters. En relación con el número de vecinos, podemos observar que cuando es pequeño estamos detectando como inliers puntos del menor de los clusters, mientras que cuanto mayor es el valor, mayor es el número de outliers detectados en dicho cluster.

Para el segundo problema, en las soluciones que obtenemos (figura 4.3, abajo) sí que podemos ver situaciones que concuerdan con el objetivo que buscábamos. La solución que más se acerca es la que tiene como valores `contamination = 0.1` y `n_neighbors = 35`, ya que se están detectando como outliers todos los puntos intermedios entre clusters y los que se encuentran en los bordes de las agrupaciones de puntos. Al igual que pasaba en el problema anterior, también podemos observar cómo el número de outliers asciende tanto al aumentar el valor de la contaminación como al hacerlo el número de vecinos.

4.2.3. Isolation Forest

En Isolation Forest, aunque tiene más parámetros, nos vamos a centrar en `contamination` y `n_estimators`, ya que al ser problemas de 2 dimensiones con pocos puntos, los otros dos parámetros afectan menos al resultado final. Para la representación de los resultados, se van a volver a exponer en forma de matriz con los valores de `contamination` (0.1, 0.3 y 0.5) creciendo en el eje

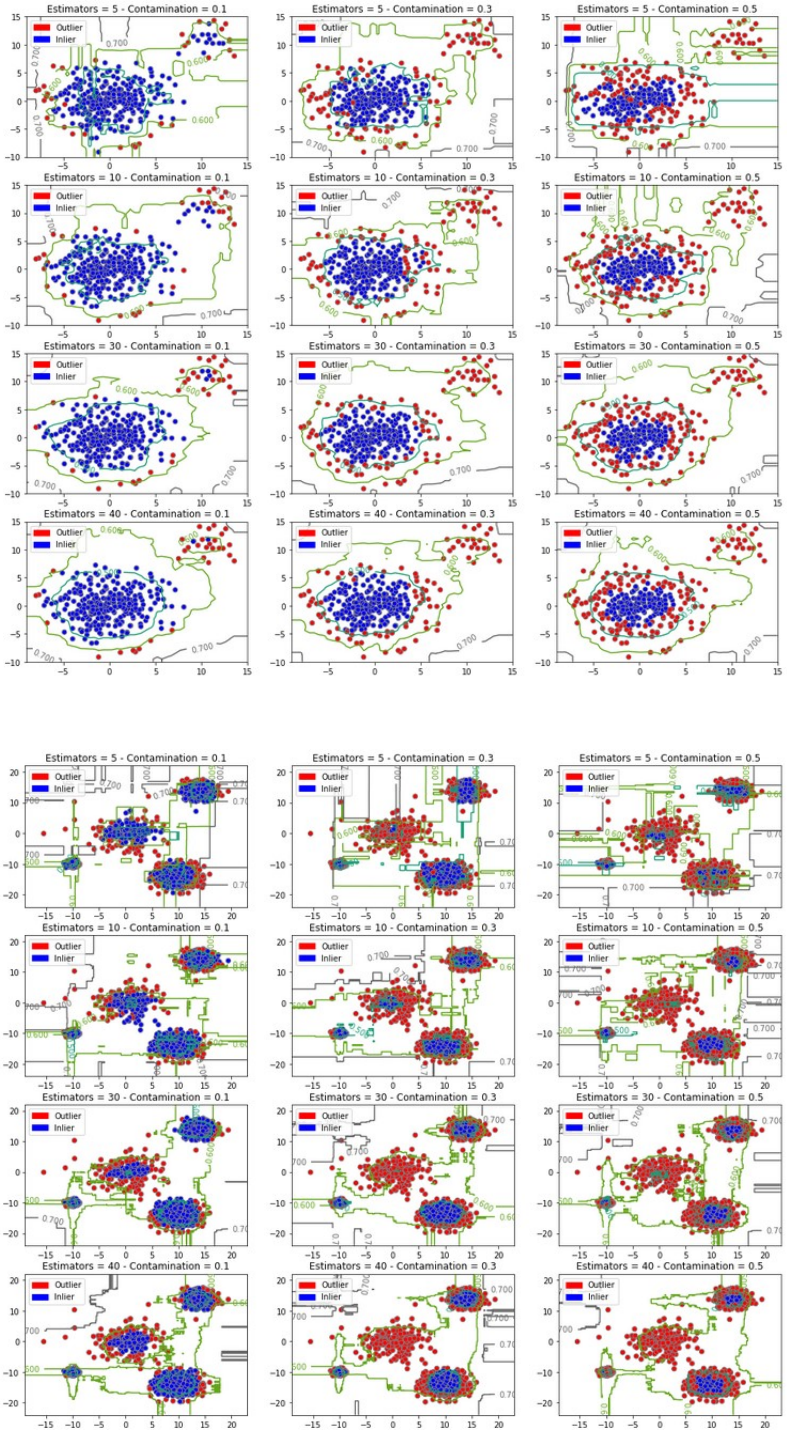

Figura 4.4: Resultados al problema sintético 1 (arriba) y al problema sintético 2 (abajo) mediante IF.

x de izquierda a derecha y los valores de `n_estimators` (5, 10, 30 y 40) creciendo en el eje y de arriba a abajo.

Para el problema 1 (figura 4.4, arriba), la mejor solución con respecto a lo esperado, se da con los valores `contamination = 0.1` y `n_estimators = 40`, aunque como podemos ver, se están detectando

dos inliers en el cluster pequeño. Como pasaba en el método anterior, ajustando un poco los parámetros podríamos evitar esto. De los resultados obtenidos, también podrían ser válidos los que tienen valores `contamination = 0.3` y `n_estimators = 30` , ya que, aunque se están detectando como outliers bastantes puntos del cluster grande, no se encuentran inliers en el superior.

Al igual que en los modelos anteriores, aumentar el parámetro `contamination` hace aumentar significativamente el número de outliers. En cambio, modificar el número de estimadores no afecta en gran medida al número de outliers detectados, aunque sí que cambia la forma de las curvas de nivel, lo que sí que afectará a la hora de detectar nuevos outliers.

En los resultados del problema 2 (figura 4.4, abajo) la solución que más se acerca a lo que buscamos, la podemos ver con `contamination = 0.1` y `n_estimators = 5`, ya que en el resto de soluciones, o se están detectando como inliers los puntos que se encuentran a medio camino entre clusters, o se están detectando demasiados outliers en los propios clusters.

4.2.4. One-class SVM

Puesto que el núcleo a utilizar es el gaussiano, las variables que vamos a estudiar son `nu` y `gamma`. Siguiendo la representación de los resultados en forma de matriz de los casos anteriores, para este modelo los valores de `nu` (0.1, 0.3 y 0.5) crecen en el eje x de izquierda a derecha y los valores de `gamma` (0.01, 0.03, 0.05 y 0.1) lo hacen en el eje y de arriba a abajo.

En el caso del problema 1 (figura 4.5, arriba), para evitar que el cluster pequeño contenga outliers tenemos que permitir que se consideren como tal un mayor número de puntos en el cluster grande. Esto hace que la mejor solución se dé con los valores `nu=0.3` y `gamma=0.01`. Como podemos observar, la variable `nu` de este modelo se comporta de manera similar a la variable `contamination` de los otros, aumentando la proporción de outliers a la vez que aumenta el valor de la misma. La variable `gamma` afecta directamente al kernel, haciendo que a mayor valor más se intente ajustar a los datos.

La solución que más se ajusta a nuestro objetivo para el problema 2 (figura 4.5, abajo) es la dada por los valores `nu = 0.1` y `gamma = 0.1`, ya que está ajustando el centro de todos los clusters y detectando como outliers los puntos más extremos de los mismos y los puntos que se encuentran entre ellos. De todas maneras, llama la atención cómo aun así se están detectando como outliers algunos de los puntos que se encuentran en el centro de los clusters más grandes. Al igual que en el problema anterior, podemos apreciar cómo aumentar el valor de `nu` implica un mayor número de outliers y aumentar el valor de `gamma` hace que se sobreajusten los datos.

4.3. Detección supervisada de outliers y metodología de evaluación

4.3.1. Motivación de un nuevo método

A la hora de hacer experimentos, en primera instancia, se intentaron optimizar los diferentes parámetros mediante el uso del método `GridSearchCV`, con el cual de forma automática se prueban diferentes configuraciones y se escoge la mejor de ellas, utilizando un proceso de validación cruzada. El problema de este método, es que a no ser que se esté probando con un conjunto de datos etiquetado específicamente para problemas de detección de outliers y anomalías, no

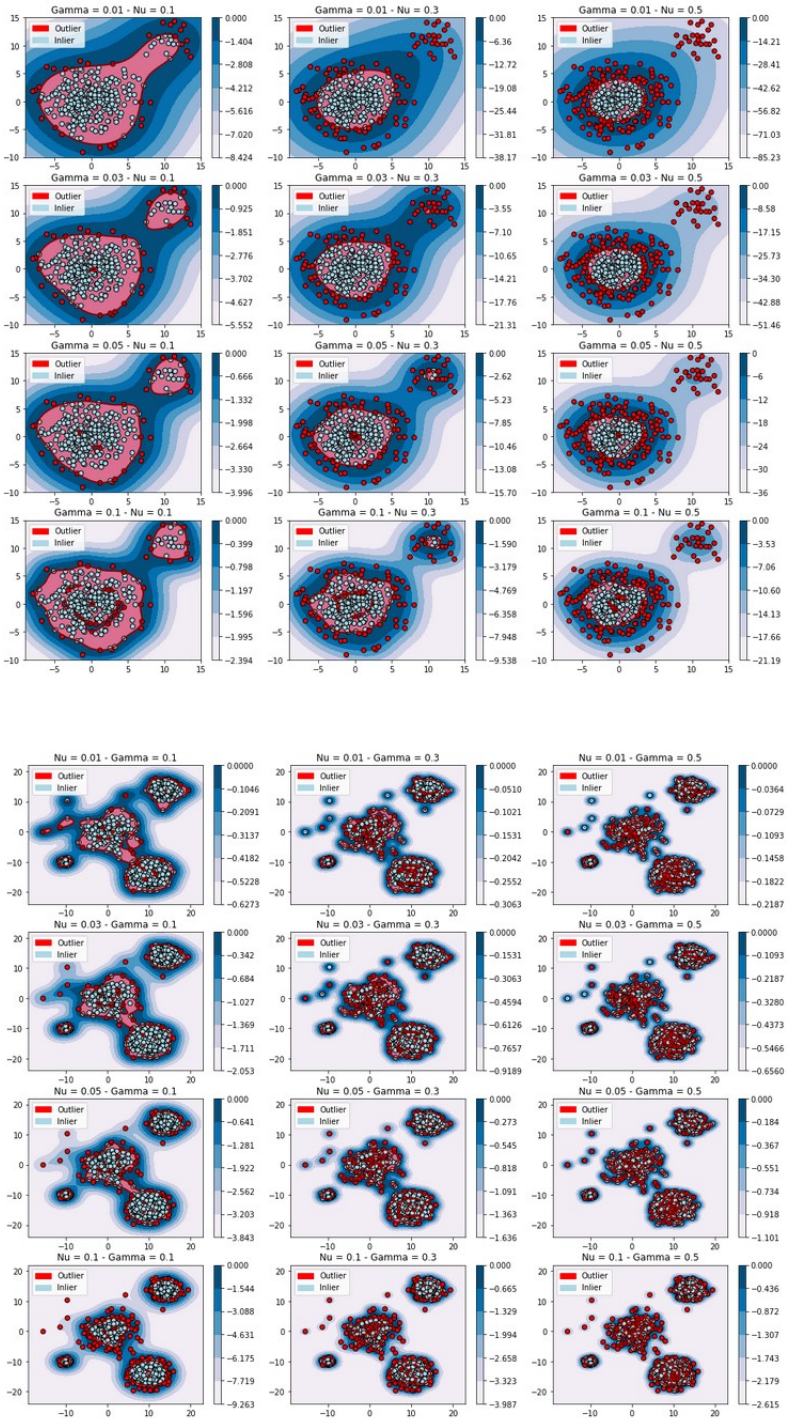

Figura 4.5: Resultados al problema sintético 1 (arriba) y al problema sintético 2 (abajo) mediante OCSVM.

disponemos de una función de score que nos permita evaluar si estamos mejorando los resultados o no. Por ejemplo, no podemos saber si se están detectando bien los outliers en un conjunto de datos de regresión como el de Boston Housing si estamos comparando las salidas de los métodos de detección de outliers (1 para inlier y -1 para outlier) con las etiquetas originales (los precios de las casas van desde $5000 hasta $50000). En la página web [15] nos podemos encontrar con

algunos conjuntos de datos adaptados a la detección de anomalías, aunque esta adaptación no siempre es muy precisa, ya que en algunos casos se etiquetan los datos como outliers en función de su clase original, no porque sean realmente datos anómalos.

Para solucionar este problema y poder hallar los mejores parámetros posibles se propone aquí un sistema basado en la concatenación de un detector de outliers y de un predictor, ya sea para problemas de clasificación o de regresión. La idea principal es que si entrenamos un modelo predictivo sin outliers, los resultados del mismo deberían ser mejores, por lo que la concatenación de un modelo de detección de outliers junto con un modelo predictivo, ambos bien parametrizados, nos debería ofrecer unos mejores resultados de predicción. De esa manera, al enfrentarnos a un problema de detección de outliers los pasos a seguir son los siguientes:

- Seleccionamos diferentes combinaciones de valores para los hiperparámetros de los modelos de detección de outliers y predicción.
- Para cada una de esas combinaciones entrenamos el modelo de detección de outliers y generamos un conjunto de datos supuestamente libre de outliers.
- Con esos subconjuntos de datos compuestos por inliers entrenamos los diferentes modelos predictivos.
- De todas la combinaciones realizadas nos quedamos con la que nos ofrezca un mejor resultado en predicción, es decir, la que obtenga una mayor puntuación dada una métrica de evaluación.

Una vez realizado esto, ya dispondremos del modelo que mejor detecta anomalías en nuestro problema. Para asegurarnos, podemos limpiar el conjunto de entrenamiento y buscar el mejor predictor posible para este conjunto de datos compuesto por inliers. Después, si evaluamos el nuevo modelo con un conjunto de test limpio de outliers y con otro sin limpiar, deberíamos obtener mejores resultados con el primero de ellos que con el segundo.

4.3.2. Implementación del nuevo método

Para recrear el comportamiento descrito anteriormente en Python, vamos a crear una clase que herede de la clase `BaseEstimator` de `scikit-learn` para poder trabajar con métodos como `GridSearchCV`. En esta nueva clase es suficiente con implementar los siguientes métodos:

- `__init__`: es el método encargado de inicializar las variables del estimador con los argumentos pasados a la hora de crearlo.
- `fit`: este método es el encargado de realizar todo el trabajo. En él, primero entrenaremos un método de detección de outliers y lo utilizaremos para limpiar la muestra de entrenamiento. Una vez hecho esto, entrenaremos el modelo de predicción con los datos definidos como inliers.
- `predict`: este método se ocupa de realizar las predicciones del conjunto de datos, por lo que utilizaremos el método `predict` del modelo de predicción sobre los datos pasados como argumento. En este caso no es necesario limpiar los datos.
- `score`: en este caso también utilizaremos el método `score` del modelo de predicción, ya que lo que nos interesa es mejorar la puntuación obtenida en predicción.

- `decision_function`: este método es necesario para poder decidir si un punto es un outlier o no, ya que devuelve un número positivo si se trata de un inlier o un numero negativo si se trata de una anomalía. Cuanto más negativo sea el valor, más anómalo será considerado un punto. Aquí utilizaremos el método `decision_function` del método de detección de outliers escogido en la creación del estimador.

En el apéndice A podemos encontrar la implemetación `Python` del estimador.

A la hora de crear un objeto de esta clase, debemos pasar como argumento una instancia de un modelo de detección de anomalías, un diccionario con los hiperparámetros de dicho detector, un predictor, ya sea para clasificación o regresión y un diccionario con los hiperparámetros de dicho predictor. La razón para tener que pasar los diferentes hiperparámetros de los modelos en forma de diccionario es utilizar esta clase de forma eficiente con el método `GridSearchCV`, independientemente de los modelos a utilizar. Si esto no se hiciese así, nos veríamos obligados a ajustar los parámetros del estimador creado a los de los modelos de detección de anomalías y predicción que se fuesen a utilizar en cada momento. Además de estos parámetros, la clase también nos permite añadir la columna target a la hora de entrenar el modelo de detección de anomalías. Esto lo hacemos porque en algunas ocasiones la condición de outlier no viene dada únicamente por los datos, sino por una combinación de los datos y la predicción. Esto puede darse, por ejemplo, en el problema de las viviendas de Boston, en el que se puede dar el caso de una vivienda con características de una casa cara, pero con un precio bajo.

Otra característica de la que dispone este estimador es el atributo `num_clean_`, que una vez entrenado el modelo nos indica el número de patrones que han sido considerados inliers, es decir, la cantidad de datos con los que se ha entrenado el predictor.

En el siguiente extracto de código podemos ver un ejemplo de utilización de esta clase.

```
lista_nu = [0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.4, 0.5]
lista_gamma = [0.01, 0.05, 0.1, 0.2, 0.5, 0.8, 1.0, 1.5, 2.0]
lista_alpha = [0.01, 0.1, 1, 10, 100, 1000, 10000, 100000]

(X_train, X_test, y_train, y_test) = train_test_split(X, y,
                                                      test_size=0.25,
                                                      shuffle=True,
                                                      random_state=10)

scl = preprocessing.MinMaxScaler()
clf_outliers = OutlierDetector(outlier_detector=OneClassSVM(),
                               model=Ridge(), add_target=False)

pipe_detector = Pipeline([('scaler', scl), ('detector', clf_outliers)])

lista_params_outlier = [{'nu': elem[0], 'gamma': elem[1]} for elem in
                        product(lista_nu, lista_gamma)]
lista_params_model = [{'alpha': elem[0]} for elem in
                      product(lista_alpha)]

param_grid = {'detector__params_outlier': lista_params_outlier,
              'detector__params_model': lista_params_model}

clf_search = GridSearchCV(pipe_detector, param_grid=param_grid,
                          scoring='neg_mean_absolute_error', n_jobs=-2,
                          verbose=0, cv=3)
clf_search.fit(X_train.values, y_train.values)

print('Cantidad de patrones detectados como inliers: {}'.format(
    clf_search.best_estimator_.named_steps['detector'].num_clean_))

pred = clean_search.best_estimator_.predict(X_test)
```

4.3.3. Metodología de evaluación

Como ya hemos comentado, la evaluación de la detección de outliers la vamos a realizar mediante la hiperparametrización de los diferentes modelos de detección de outliers junto con un modelo de predicción, ya sea para clasificación o para regresión. Esto lo hacemos con el fin de observar si entrenando con el conjunto de datos libre de outliers obtenemos un mejor resultado de predicción. En el caso de problemas de clasificación, como modelo predictor vamos a utilizar una regresión logística. Por otro lado, al trabajar con problemas de regresión usaremos un modelo Ridge.

Con el fin de obtener unos resultados más fiables, vamos a repetir el proceso de evaluación un total de 10 veces para cada modelo con conjuntos de train y test distintos obtenidos mediante `ShuffleSplit`. Finalmente promediaremos los distintos resultados obtenidos. Los datos que se van a promediar son:

- Número de patrones detectados como inliers en el mejor modelo seleccionado por `GridSearchCV`.
- Puntuaciones obtenidas con las diferentes métricas con el conjunto de test completo.
- Puntuaciones obtenidas con las diferentes métricas con el conjunto de test formado únicamente por inliers.

Además, vamos a extraer los índices de los datos detectados como inliers y como outliers en el proceso de entrenamiento. Esto nos permitirá después extraer información de la distribución de los datos. Para ello representaremos los datos en diagramas de cajas o boxplots en función de las diferentes variables. En el caso de problemas de clasificación también haremos distinción en función de la clase.

Puesto que la división en subconjuntos de entrenamiento y test se está haciendo mediante `ShuffleSplit` hay solapamiento entre las diferentes iteraciones, por lo que a la hora de extraer los índices de los puntos detectados como inliers y outliers puede darse el caso de que un mismo patrón sea detectado como inlier en una iteración y como outlier en otra.

Evaluación de los métodos de detección de outliers

Lo primero que vamos a hacer es dividir de forma aleatoria el conjunto original en dos subconjuntos, uno formado por tres cuartas partes de los datos, que representará el conjunto de entrenamiento y validación de los modelos, y otro formado por la cuarta parte de los datos restantes, que utilizaremos para realizar tests y evaluar la eficiencia del proceso mediante diferentes métricas.

Después de esto se va a hiperparametrizar el estimador formado por el detector de outliers y el predictor mediante el uso del método `GridSearchCV`. En el caso de MCD y LOF, antes del estimador se va a utilizar el estandarizado `StandardScaler` para que todas las variables tengan media 0 y desviación típica 1. Con el modelo de One-class SVM vamos a utilizar el transformador `MinMaxScaler` para que todas las variables estén en el rango [0,1]. En el caso de Isolation Forests no es necesario realizar este tipo de preprocesamiento ya que está basado en árboles de decisión, por lo que la diferencia de órdenes de las variables no afecta al resultado.

A la hora de configurar la hiperparametrización con `GridSearchCV`, la división del conjunto de entrenamiento en varias particiones se va a realizar mediante la utilización del método

`StratifiedKFold` en el caso de clasificación y `KFold` en el caso de regresión; en ambos casos con un `n_folds = 3`, es decir, se va a dividir el conjunto de entrenamiento en tres subconjuntos de entrenamiento/validación sin solapamiento. Para la evaluación de las diferentes combinaciones y selección de la mejor de ellas, se va a utilizar la métrica del área bajo la curva precision-recall (AP) (`average_precision`) en el caso de clasificación y la métrica del error absoluto medio (MAE) (`neg_mean_absolute_error`) en el caso de regresión.

Una vez completado este proceso con el cual hemos obtenido el estimador con mejor puntuación, utilizaremos el detector de outliers para limpiar tanto el conjunto de entrenamiento como el de test. Estos conjuntos son los que utilizaremos a continuación para comprobar si estamos mejorando la predicción original o no.

Para ello, lo que vamos a hacer es hiperparametrizar un modelo de predicción mediante el uso del conjunto de entrenamiento limpio obtenido anteriormente. Una vez hecho esto, utilizaremos tanto el conjunto de test sin limpiar, es decir, con outliers e inliers mezclados, como el conjunto de test limpio para realizar predicciones y obtener diferentes métricas de evaluación. Con esto, deberíamos obtener unos resultados mejores en los test formados únicamente por inliers que cuando estos se mezclan con outliers. De la misma manera, estos resultados deberían ser mejores que si no hacemos todo el proceso de detección y supresión de anomalías. Las métricas a comparar en el caso de clasificación son Accuracy, Precision, Recall, área bajo la curva ROC (AUC) y área bajo la curva precision-recall (AP), mientras que en regresión utilizaremos el error absoluto medio (MAE), el error cuadrático medio (MSE) y el coeficiente R2.

A continuación podemos ver el pseudocódigo del proceso de evaluación.

for i=0 to 10 **do**
    $train\_full, test\_full \leftarrow$ División de los datos en entrenamiento y test
    $best\_model \leftarrow$ Hiperparametrización mediante GridSearchCV con $train\_full$
    Obtención del número de datos con los que se ha entrenado el modelo de predicción
    Obtención de la puntuación del mejor modelo
    $train\_clean, test\_clean \leftarrow$ Utilización del $best\_model$ para limpiar $train\_full$ y $test\_full$
    $model\_inliers \leftarrow$ Hiperparametrización de un modelo predictor con $train\_clean$
    $pred\_full \leftarrow$ Realizar predicciones con $model\_inliers$ sobre $test\_full$
    Obtención de las diferentes métricas a partir de $pred\_full$
    $pred\_clean \leftarrow$ Realizar predicciones con $model\_inliers$ sobre $test\_clean$
    Obtención de las diferentes métricas a partir de $pred\_clean$
end for

Evaluación mediante votación

A la hora de realizar experimentos con conjuntos de datos reales, además de los métodos de detección de outliers vistos en el capítulo 3 vamos a utilizar un método de votación. En este caso, en lugar de realizar el proceso de hiperparametrización descrito anteriormente, cada uno de los modelos obtenidos mediante `GridSearchCV` votará sobre si un punto es un outlier o un inlier. Después de esto configuraremos el conjunto de datos limpios de tal manera que se encuentren en él aquellos puntos que han sido detectados como inliers por todos los métodos y aquellos que hayan sido detectados como outliers por un único método. Una vez obtenidos estos datos, podemos seguir la evaluación igual que en el resto de métodos, entrenando un modelo predictor con datos limpios y evaluándolo con un conjunto de test con outliers y con un conjunto de test formado únicamente por inliers. Este conjunto de test se va a limpiar de la misma manera que

	count	mean	std	min	25 %	50 %	75 %	max
Pregnancies	768.00	3.85	3.37	0.00	1.00	3.00	6.00	17.00
Glucose	768.00	120.89	31.97	0.00	99.00	117.00	140.25	199.00
BloodPresure	768.00	69.11	19.36	0.00	62.00	72.00	80.00	122.00
SkinThickness	768.00	20.54	15.95	0.00	0.00	23.00	32.00	99.00
Insulin	768.00	79.80	115.24	0.00	0.00	30.50	127.25	846.00
BMI	768.00	31.99	7.88	0.00	27.30	32.00	36.60	67.10
DiabetesPedigreeFunction	768.00	0.47	0.33	0.08	0.24	0.37	0.63	2.42
Age	768.00	33.24	11.76	21.00	24.00	29.00	41.00	81.00
Target	768.00	0.35	0.48	0.00	0.00	0.00	1.00	1.00

Tabla 4.1: Descripción del problema de la diabetes de los indios Pima.

se ha limpiado el de entrenamiento.

4.4. Detección de outliers en el problema de la diabetes de los indios Pima

4.4.1. Descripción del problema y los datos

Para comprobar el funcionamiento y la eficacia de los diferentes métodos de detección de outliers y anomalías en una muestra real para un problema de clasificación vamos a utilizar los datos de Pima indians diabetes, creado por el National Institute of Diabetes and Digestive and Kidney Diseases. Se trata de un problema en el que se predice si los pacientes (768 mujeres de al menos 21 años pertenecientes a la etnia Pima) tienen diabetes o no. Para ello se utilizan las siguientes variables [16]:

- Pregnancies: número de veces que se ha estado embarazada.
- Glucose: Concentración de glucosa plasmática a las 2 horas de una prueba de tolerancia oral a la glucosa (mg/dl).
- BloodPressure: Presión arterial diastólica (mmHg).
- SkinThickness: Grosor del pliegue de la piel del tríceps (mm).
- Insulin: Concentración de insulina sérica a las 2 horas de una prueba de tolerancia oral a la glucosa (mU/ml).
- BMI: Índice de masa corporal (peso /altura al cuadrado en kg/m^2).
- DiabetesPedigreeFunction: Función de pedigrí de la diabetes.
- Age: edad (años).

En la tabla 4.1 podemos ver una descripción de los datos.

Si observamos la tabla lo primero que nos debe llamar la atención es que los atributos Glucose, BloodPressure, SkinThickness, Insulin y BMI tienen como valor mínimo 0, cosa que no tiene

	Base
Conjunto de test	Completo
Accuracy	0.763
Precision	0.714
Recall	0.548
Roc AUC	0.713
AP	0.548

Tabla 4.2: Resultados en test sin detección de outliers en Pima.

sentido en seres vivos. Por otro lado, también nos puede llamar la atención que en los atributos SkinThickness, Insulin y DiabetesPedigreeFunction el valor máximo es muy superior a la mayoría de los datos. Estos valores extremos son los que nos pueden indicar una presencia de patrones anómalos, por lo que deberemos fijarnos en ellos.

Para establecer un punto de partida, siguiendo la metodología de evaluación explicada en 4.3.3, hemos utilizado el método `GridSearchCV` con AP como métrica de scoring para encontrar la mejor regresión logística utilizando el conjunto de datos completos, es decir, sin detección de outliers. Para ello hemos probado diferentes valores del parámetro `C` (0.01,0.1,1,10,100,1000,10000,100000). Con este modelo, tras las 10 iteraciones, hemos conseguido una puntuación AP media de 0.724±0.021 en entrenamiento. En la tabla 4.2 podemos ver los resultados obtenidos en test.

4.4.2. Detección mediante MCD

Hiperparametrización

Para hiperparametrizar este modelo se ha utilizado el método `GridSearchCV` con AP como métrica de scoring y los diferentes conjuntos de entrenamiento y validación obtenidos mediante `ShuffleSplit` como se ha explicado en 4.3.3 junto con la siguiente rejilla de valores para los diferentes parámetros:

- `contamination`: `[0.05,0.1,0.15,0.2,0.25,0.3,0.4,0.5]`.
- `support_fraction`: `[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9]`.
- `C`: `[0.01,0.1,1,10,100,1000,10000,100000]`.

De los 576 patrones utilizados durante el proceso de hiperparametrización del detector de anomalías, se han detectado como outliers un promedio de 89±82 puntos, por lo que el modelo de predicción se ha entrenado con una media de 486±82 puntos. En este caso hay que destacar que la desviación típica de los outliers es muy similar a la media, lo que nos indica que en alguna de las 10 iteraciones no se han encontrado casi anomalías mientras que en otras se han encontrado más de 150. Esto nos puede indicar que este método no es muy estable.

	Base	MCD	
Conjunto de test	Completo	Completo	Inliers
Accuracy	0.763	0.756	0.779
Precision	0.714	0.684	0.718
Recall	0.548	0.569	0.508
Roc AUC	0.713	0.713	0.703
AP	0.548	0.540	0.517

Tabla 4.3: Resultados en test con MCD en Pima.

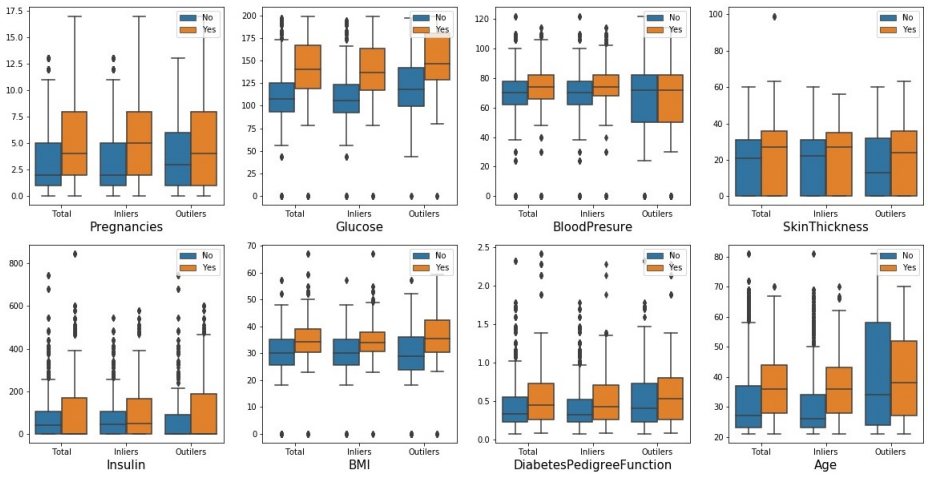

Figura 4.6: Diagrama de cajas obtenido mediante MCD en Pima.

Resultados en detección de outliers y anomalías

Como se explica en la sección 4.3.3, tras la creación de un conjunto de datos formado únicamente por los puntos detectados como inliers tras la etapa de hiperparametrización, se ha entrenado una regresión logística con dichos datos y se ha evaluado su eficiencia mediante el uso de un conjunto de test, tanto manteniéndolo intacto como limpiándolo de outliers. En la tabla 4.3 se muestran los resultados obtenidos para las diferentes métricas, de media, en las 10 iteraciones realizadas.

Como podemos observar, en términos de AP no estamos obteniendo una mejora con respecto al modelo base, ni con el conjunto de test completo ni con únicamente los inliers. En cambio, sí que podemos ver cierta mejoría en Accuracy y Precision con el conjunto de test de inliers y en Recall en el conjunto de test completo.

Además de esto, hemos obtenido unos diagramas de cajas de las diferentes variables a partir de los datos de entrenamiento con el fin de observar la distribución de los datos en función de si son inliers o outliers. En la figura 4.6 podemos ver, de izquierda a derecha, la distribución del total de datos de entrenamiento como referencia, la de los clasificados como inliers y la de los clasificados como outliers, en todas ellas haciendo distinción en la clase.

Si nos fijamos en la mediana de las cajas de los outliers, las variables que más llaman la atención son SkinThickness y Age en el caso no tener diabetes e Insulin tanto en el caso de tener diabetes como en el caso de no tenerla. En función del rango intercuartílico, las variables que más destacan

	Base	MCD		LOF	
Conjunto de test	Completo	Completo	Inliers	Completo	Inliers
Accuracy	0.763	0.756	0.779	0.764	0.767
Precision	0.714	0.684	0.718	0.707	0.712
Recall	0.548	0.569	0.508	0.561	0.550
Roc AUC	0.713	0.713	0.703	0.717	0.715
AP	0.548	0.540	0.517	0.550	0.546

Tabla 4.4: Resultados en test con LOF en Pima.

son BloodPresure y Age.

4.4.3. Detección mediante LOF

Hiperparametrización

Al igual que en el caso anterior, siguiendo la metodología de evaluación del capítulo 4.3.3, vamos a hiperparametrizar este modelo utilizando el método `GridSearchCV` con AP como métrica de scoring y los diferentes conjuntos de entrenamiento y validación obtenidos mediante `ShuffleSplit` como se ha explicado en 4.3.3 junto con la siguiente rejilla de valores para los diferentes parámetros:

- `contamination`: `[0.05,0.1,0.15,0.2,0.25,0.3,0.4,0.5]`.
- `n_neighbors`: `[5, 9, 17, 33, 65]`($1+2^k$ con k=2,3,4,5,6).
- `C`: `[0.01,0.1,1,10,100,1000,10000,100000]`.

En esta ocasión, de los 576 patrones utilizados durante el proceso de hiperparametrización del detector de anomalías, se han detectado como outliers un promedio de 21±3 puntos, por lo que el modelo de predicción se ha entrenado con una media de 555±3 puntos.

Resultados en detección de outliers y anomalías

Como en el caso anterior, hemos entrenado una regresión logística con los datos detectados como inliers en el conjunto de entrenamiento y hemos realizado la evaluación sobre un conjunto de test, tanto manteniéndolo intacto como limpiándolo de outliers. En la tabla 4.4 se muestran los resultados obtenidos para las diferentes métricas, de media, en las 10 iteraciones realizadas.

En esta ocasión, sí que hemos conseguido una ligera mejoría en AP con respecto al modelo base con el conjunto de test completo y una puntuación muy similar en el caso del conjunto de test con inliers. En el resto de métricas, excepto en Precision, podemos ver un incremento en las puntuaciones, tanto con los datos completos como con los inliers.

De nuevo, hemos obtenido unos diagramas de cajas de las diferentes variables a partir de los datos de entrenamiento con el fin de observar la distribución de los datos en función de si son inliers o outliers. En la figura 4.7 podemos ver, de izquierda a derecha, la distribución del total de datos de entrenamiento como referencia, la de los clasificados como inliers y la de los clasificados como outliers, en todas ellas haciendo distinción en la clase.

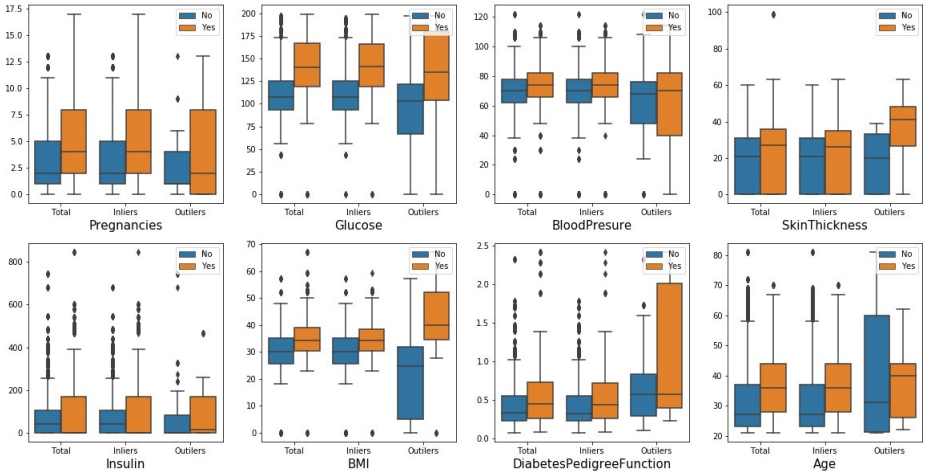

Figura 4.7: Diagrama de cajas obtenido mediante LOF en Pima.

En términos de la mediana de los outliers, las variables más llamativas son SkinThickness cuando se tiene diabetes, Insulin cuando no se tiene y BMI en ambos casos. Si nos fijamos en el rango intercuartílico de los outliers, las variables que presentan mayores diferencias son BloodPresure, SkinThickness y DiabetesPedigreeFunction cuando se tiene diabetes y Age cuando no se tiene.

4.4.4. Detección mediante Isolation Forest

Hiperparametrización

Al igual que para los métodos anteriores, vamos a hiperparametrizar este modelo utilizando el método `GridSearchCV` con AP como métrica de scoring y los diferentes conjuntos de entrenamiento y validación obtenidos mediante `ShuffleSplit` como se ha explicado en 4.3.3 junto con la siguiente rejilla de valores para los diferentes parámetros:

- `contamination`: `[0.05,0.1,0.15,0.2,0.25,0.3,0.4,0.5]`.
- `n_estimators`: `[10,25,50,75,100,125]`.
- `max_features`: `[0.125,0.25,0.375,0.5,0.625,0.77,0.875,1.0]`.
- `C`: `[0.01,0.1,1,10,100,1000,10000,100000]`.

En este modelo, de los 576 patrones utilizados durante el proceso de hiperparametrización del detector de anomalías, se han detectado como outliers un promedio de 150±75 puntos, por lo que el modelo de predicción se ha entrenado con una media de 426±75 puntos. Al igual que pasaba en MCD, la desviación típica del número de outliers es bastante alta, aunque en este caso no está tan cerca de la media.

Resultados en detección de outliers y anomalías

Como en los casos anteriores, siguiendo la metodología de evaluación del capítulo 4.3.3, hemos entrenado una regresión logística con los datos detectados como inliers en el conjunto de

	Base	MCD		LOF		IF	
Conjunto de test	Completo	Completo	Inliers	Completo	Inliers	Completo	Inliers
Accuracy	0.763	0.756	0.779	0.764	0.767	0.758	0.761
Precision	0.714	0.684	0.718	0.707	0.712	0.696	0.703
Recall	0.548	0.569	0.508	0.561	0.550	0.554	0.473
Roc AUC	0.713	0.713	0.703	0.717	0.715	0.710	0.687
AP	0.548	0.540	0.517	0.550	0.546	0.541	0.507

Tabla 4.5: Resultados en test con Isolation Forest en Pima.

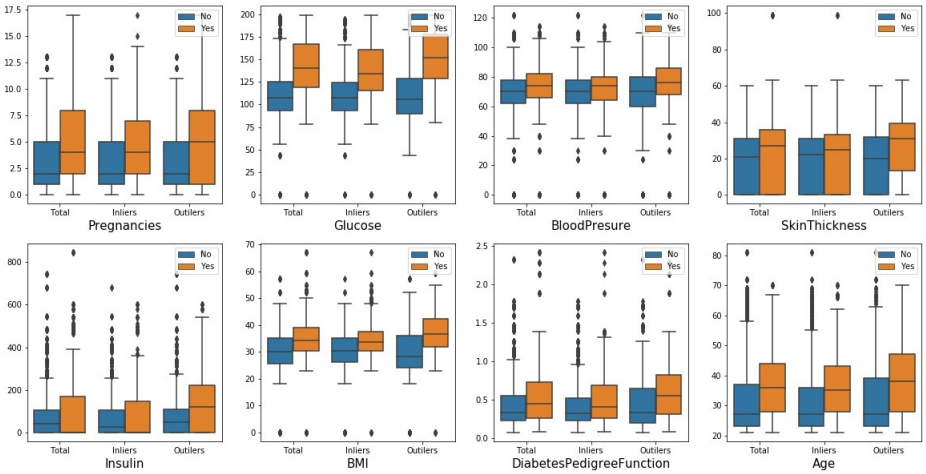

Figura 4.8: Diagrama de cajas obtenido mediante Isolation Forest en Pima.

entrenamiento y hemos realizado la evaluación sobre un conjunto de test, tanto manteniéndolo intacto como limpiándolo de outliers. En la tabla 4.5 se muestran los resultados obtenidos para las diferentes métricas, de media, en las 10 iteraciones realizadas.

Con este modelo nos vuelve a pasar que en AP no conseguimos mejorar los resultados del modelo base, ni con los datos de test completos ni con los inliers. Con el resto de medidas pasa algo similar, no se consigue una mejoría con respecto al modelo base. La única métrica que sí lo hace es Recall con los datos completos, aunque, de todas maneras, sigue siendo menor que la obtenida con LOF en los datos completos.

De nuevo, hemos obtenido unos diagramas de cajas de las diferentes variables a partir de los datos de entrenamiento con el fin de observar la distribución de los datos en función de si son inliers o outliers. En la figura 4.8 podemos ver, de izquierda a derecha, la distribución del total de datos de entrenamiento como referencia, la de los clasificados como inliers y la de los clasificados como outliers, en todas ellas haciendo distinción en la clase.

En esta ocasión, las medianas de los outliers que más llaman la atención son las del las variables Insulin y DiabetesPedigreeFunction cuando se tiene diabetes. Con respecto a los rangos intercuartílicos, la única variable que presenta una diferencia notable es SkinThickness cuando se tiene diabetes.

	Base	MCD		LOF		IF		One-class	
Conjunto de test	Completo	Completo	Inliers	Completo	Inliers	Completo	Inliers	Completo	Inliers
Accuracy	0.763	0.756	0.779	0.764	0.767	0.758	0.761	0.764	0.777
Precision	0.714	0.684	0.718	0.707	0.712	0.696	0.703	0.692	0.715
Recall	0.548	0.569	0.508	0.561	0.550	0.554	0.473	0.588	0.534
Roc AUC	0.713	0.713	0.703	0.717	0.715	0.710	0.687	0.723	0.714
AP	0.548	0.540	0.517	0.550	0.546	0.541	0.507	0.550	0.534

Tabla 4.6: Resultados en test con One-class SVM en Pima.

4.4.5. Detección mediante One-Class SVM

Hiperparametrización

Al igual que para los métodos anteriores, siguiendo la metodología de evaluación del capítulo 4.3.3, vamos a hiperparametrizar este modelo utilizando el método `GridSearchCV` con AP como métrica de scoring y los diferentes conjuntos de entrenamiento y validación obtenidos mediante `ShuffleSplit` como se ha explicado en 4.3.3 junto con la siguiente rejilla de valores para los diferentes parámetros:

- `nu`: `[0.05,0.1,0.15,0.2,0.25,0.3,0.4,0.5]`.
- `gamma`: `[0.02,0.24,0.46,0.68,0.90,1.12,1.34,1.56,1.78,2.00]`.
- `C`: `[0.01,0.1,1,10,100,1000,10000,100000]`.

En este modelo, de los 576 patrones utilizados durante el proceso de hiperparametrización del detector de anomalías, se han detectado como outliers un promedio de 98±26 puntos, por lo que el modelo de predicción se ha entrenado con una media de 478±26 puntos. En este método volvemos a ver una desviación típica de los outliers llamativa, aunque como en el método anterior, no tan cercana a la media como sucedía en MCD.

Resultados en detección de outliers y anomalías

Como en los casos anteriores, hemos entrenado una regresión logística con los datos detectados como inliers en el conjunto de entrenamiento y hemos realizado la evaluación sobre un conjunto de test, tanto manteniéndolo intacto como limpiándolo de outliers. En la tabla 4.6 se muestran los resultados obtenidos para las diferentes métricas, de media, en las 10 iteraciones realizadas.

En esta ocasión, volvemos a ver una mejoría en AP con respecto al modelo base tanto en el conjunto de test completo como el de los inliers. De todas maneras, estas puntuaciones no superan a las obtenidas con LOF, que, hasta el momento, son las mejores. En cambio, en Accuracy, Precision, ROC y Recall con los datos completos sí que estamos consiguiendo una mejoría con respecto al modelo base y a LOF.

De nuevo, hemos obtenido unos diagramas de cajas de las diferentes variables a partir de los datos de entrenamiento con el fin de observar la distribución de los datos en función de si son inliers o outliers. En la figura 4.9 podemos ver, de izquierda a derecha, la distribución del total de datos de entrenamiento como referencia, la de los clasificados como inliers y la de los clasificados como outliers, en todas ellas haciendo distinción en la clase.

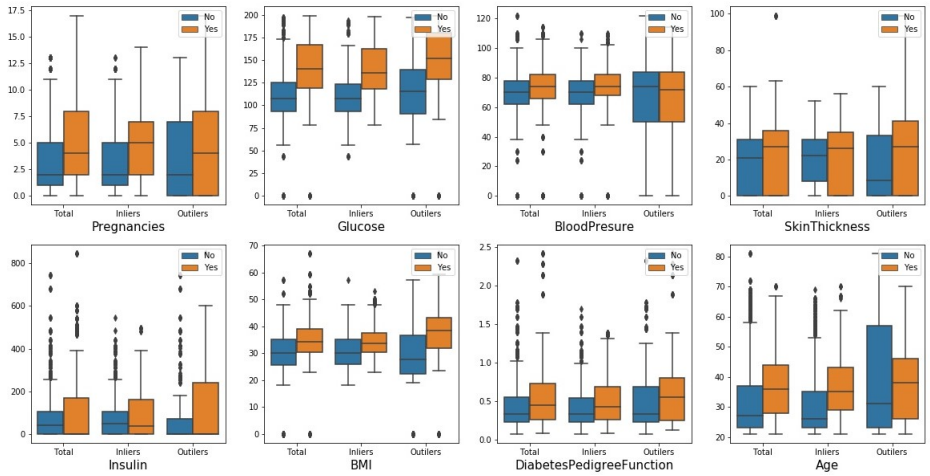

Figura 4.9: Diagrama de cajas obtenido mediante One-Class SVM en Pima.

	MCD	LOF	iForest	One-class
MCD	89.500	13.300	43.800	47.700
LOF	13.300	20.100	13.400	17.600
iForest	43.800	13.400	149.700	48.700
One-class	47.700	17.600	48.700	98.000

Tabla 4.7: Coincidencia de outliers entre métodos en Pima.

Coincidencias	0	1	2	3	4
Puntos	352.7	131.1	59.1	24.4	8.7

Tabla 4.8: Número medio de veces que un punto ha sido detectado como outlier en Pima.

Las medianas de los outliers que más llaman la atención en esta ocasión son las de las variables SkinThickness y Age cuando no se tiene diabetes e Insulin cuando se tiene y cuando no. De los rangos intercuartílicos, los más llamativos son los de BloodPresure y los de SkinThikcness y Age cuando no se tiene diabetes.

4.4.6. Detección mediante votación

Una vez analizados los outliers detectados por cada uno de los métodos por separado, vamos a analizar estas detecciones de manera conjunta. Para ello, en primer lugar vamos a comprobar las coincidencia de puntos detectados como outliers entre los diferentes métodos. En la tabla 4.7 podemos ver la cantidad de coincidencias que hay de media entre los diferentes métodos.

Después de esto, nos puede interesar simular un sistema de detección de outliers por votación. De esta manera, cada método decidirá si un punto es un outlier o no, para finalmente seleccionar como tal los que han sido clasificados como anomalías por varios métodos, como se explica en la sección 4.3.3. En la tabla 4.8 podemos ver por cuántos modelos han sido detectados como outliers, de media, los diferentes patrones del conjunto de entrenamiento.

Ahora, podemos crear un nuevo conjunto de datos en el que solo consideramos como outliers

	Base	MCD		LOF		IF		One-class		Votación	
Conjunto de test	Completo	Completo	Inliers	Completo	Inliers	Completo	Inliers	Completo	Inliers	Completo	Inliers
Accuracy	0.763	0.756	0.779	0.764	0.767	0.758	0.761	0.764	0.777	0.768	0.783
Precision	0.714	0.684	0.718	0.707	0.712	0.696	0.703	0.692	0.715	0.702	0.745
Recall	0.548	0.569	0.508	0.561	0.550	0.554	0.473	0.588	0.534	0.587	0.520
Roc AUC	0.713	0.713	0.703	0.717	0.715	0.710	0.687	0.723	0.714	0.726	0.715
AP	0.548	0.540	0.517	0.550	0.546	0.541	0.507	0.550	0.534	0.556	0.541

Tabla 4.9: Resultados en test con votación en Pima.

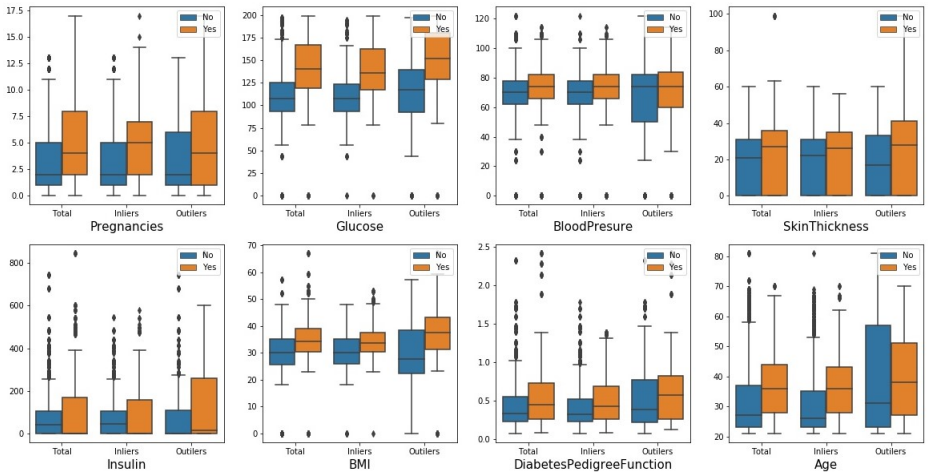

Figura 4.10: Diagrama de cajas obtenido mediante votación en Pima.

aquellos puntos que han sido detectados por dos o más métodos, mientras que, los que solo lo han sido por uno los vamos a considerar como inliers.

Como en los casos anteriores, hemos entrenado una regresión logística con los datos detectados como inliers en el conjunto de entrenamiento y hemos realizado la evaluación sobre un conjunto de test, tanto manteniéndolo intacto como limpiándolo de outliers. En la tabla 4.9 se muestran los resultados obtenidos para las diferentes métricas, de media, en las 10 iteraciones realizadas.

Con este modelo de votación podemos ver cómo la puntuación en AP con respecto al modelo base mejora con los datos de test completos y empeora ligeramente con los datos de test de inliers. En el resto de métricas se aprecian mejoras en el caso de Accuaracy, ROC, Precision en test con inliers y Recall en test completo.

Al igual que en el resto de métodos, hemos obtenido unos diagramas de cajas de las diferentes variables a partir de los datos de entrenamiento con el fin de observar la distribución de los datos en función de si son inliers o outliers. En la figura 4.10 podemos ver, de izquierda a derecha, la distribución del total de datos de entrenamiento como referencia, la de los clasificados como inliers y la de los clasificados como outliers, en todas ellas haciendo distinción en la clase.

En este caso, las medianas de los outliers presentan diferencias en las variables Glucose y SkinThickness, Insulin y Age cuando no se tiene diabetes. Con respecto a los rangos intercuartílicos de los outliers resultan llamativas las variables BloodPresure, Age e Insulin cuando se tiene diabetes.

	count	mean	std	min	25 %	50 %	75 %	max
CRIM	506.000	3.614	8.602	0.006	0.082	0.257	3.677	88.976
ZN	506.000	11.364	23.322	0.000	0.000	0.000	12.500	100.000
INDUS	506.000	11.137	6.860	0.460	5.190	9.690	18.100	27.740
CHAS	506.000	0.069	0.254	0.000	0.000	0.000	0.000	1.000
NOX	506.000	0.555	0.116	0.385	0.449	0.538	0.624	0.871
RM	506.000	6.285	0.703	3.561	5.886	6.208	6.623	8.780
AGE	506.000	68.575	28.149	2.900	45.025	77.500	94.075	100.000
DIS	506.000	3.795	2.106	1.130	2.100	3.207	5.188	12.127
RAD	506.000	9.549	8.707	1.000	4.000	5.000	24.000	24.000
TAX	506.000	408.237	168.537	187.000	279.000	330.000	666.000	711.000
PTRATIO	506.000	18.456	2.165	12.600	17.400	19.050	20.200	22.000
B	506.000	356.674	91.295	0.320	375.377	391.440	396.225	396.900
LSTAT	506.000	12.653	7.141	1.730	6.950	11.360	16.955	37.970
Target	506.000	22.533	9.197	5.000	17.025	21.200	25.000	50.000

Tabla 4.10: Descripción del problema de las viviendas de Boston.

4.4.7. Conclusiones

Como hemos podido observar, en términos de AP no estamos consiguiendo una mejora cuando realizamos los test únicamente con inliers, aunque con LOF nos quedamos bastante cerca. Si nos fijamos en otra métricas, sí que podemos ver cierta mejoría, como es el caso de accuracy en MCD, LOF, One-class y votación o precision en MCD, One-class y votación.

De todas maneras, aunque los datos numéricos no sean especialmente buenos, sí que podemos ver en los diagramas de cajas cómo los diferentes modelos están aprendiendo a diferenciar outliers de inliers. Esto lo podemos ver especialmente en las variables Insulin, SkinThickness, BloodPressure y Age.

4.5. Detección de outliers en el problema de las viviendas de Boston

4.5.1. Descripción del problema y los datos

En este caso, vamos a evaluar los diferentes métodos pero con un problema de regresión. El problema a utilizar se trata del de las viviendas de diversas zonas de Boston, disponible en `scikit-learn` y propuesto originalmente en [17]. Con este conjunto de datos, compuesto por 506 casas, tratamos de averiguar el valor medio de los precios para cada zona en función de las siguientes variables:

- CRIM: tasa de delincuencia per cápita por ciudad.
- ZN: proporción de terreno residencial dividido en zonas para lotes de más de 25000 pies cuadrados.
- INDUS: proporción de acres de negocios no minoristas por ciudad.

	Base
Conjunto de test	Completo
MAE	3.458
MSE	24.779
R2	0.711

Tabla 4.11: Resultados en test sin detección de outliers en Boston.

- CHAS: Variable dummy del río Charles (1 si el tramo limita con el río; 0 en caso contrario).
- NOX: concentración de óxidos nítricos (partes por 10 millones).
- RM: número medio de habitaciones por vivienda .
- AGE: proporción de unidades ocupadas por sus propietarios construidas antes de 1940.
- DIS: distancias ponderadas a cinco centros de empleo de Boston.
- RAD: índice de accesibilidad a carreteras radiales.
- TAX: tasa del impuesto sobre el valor total de la propiedad por cada $10000.
- PTRATIO: proporción de alumnos por docente por ciudad.
- B: $1000(Bk - 0{,}63)^2$ donde Bk es la proporción de individuos de color por ciudad.
- LSTAT: valores porcentuales de la población de estado inferior.
- Target: valor medio de las viviendas ocupadas por sus propietarios en miles de dólares.

En la tabla 4.10 podemos ver una descripción de los datos. Si observamos la tabla, nos pueden llamar la atención las variables `CRIM` y `ZN` ya que sus valores máximos son mucho mayores que la mayoría del resto de datos. También nos pueden llamar la atención las variables `TAX` y `B`, cuyas desviaciones típicas son muy altas, lo que nos indica que los valores de estas fluctúan mucho. Aunque en [17] no se menciona, existe la posibilidad de que la variable `Target` tenga como valor máximo $50.000 aunque el valor real sea mayor, ya que en la muestra podemos encontrar 15 casos con dicho valor exacto, mientras que el resto de instancias se encuentran redondeadas en las centenas.

Para establecer un punto de partida, siguiendo la metodología de evaluación explicada en 4.3.3, hemos utilizado el método `GridSearchCV` con MAE como métrica de scoring para encontrar el mejor modelo ridge utilizando el conjunto de datos completos, es decir, sin detección de outliers. Para ello hemos probado diferentes valores del parámetro `alpha` (0.01,0.1,1,10,100,1000,10000,100000). Con este modelo, tras las 10 iteraciones, hemos conseguido una puntuación MAE media de 3.454±0.100 en entrenamiento. En la tabla 4.11 podemos observar los resultados obtenidos en test.

	Base	MCD	
Conjunto de test	Completo	Completo	Inliers
MAE	3.458	4.080	2.545
MSE	24.779	50.035	11.212
R2	0.711	0.406	0.830

Tabla 4.12: Resultados en test con MCD en Boston.

4.5.2. Detección mediante MCD

Hiperparametrización

Para hiperparametrizar este modelo se ha utilizado el método `GridSearchCV` con MAE como métrica de scoring y los diferentes conjuntos de entrenamiento y validación obtenidos mediante `ShuffleSplit` como se ha explicado en 4.3.3 junto con la siguiente rejilla de valores para los diferentes parámetros:

- `contamination`: `[0.05,0.1,0.15,0.2,0.25,0.3,0.4,0.5]`.
- `support_fraction`: `[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9]`.
- `alpha`: `[0.01,0.1,1,10,100,1000,10000,100000]`.

De los 379 patrones utilizados durante el proceso de hiperparametrización del detector de anomalías, se han detectado como outliers un promedio de 67±23 puntos, por lo que el modelo de predicción se ha entrenado con una media de 312±23 puntos.

Resultados en detección de outliers y anomalías

Como se explica en la sección 4.3.3, tras la creación de un conjunto de datos formado únicamente por los puntos detectados como inliers tras la etapa de hiperparametrización, se ha entrenado una regresión ridge con ellos y se ha evaluado su eficiencia mediante el uso de un conjunto de test, tanto manteniéndolo intacto como limpiándolo de outliers. En la tabla 4.12 se muestran los resultados obtenidos para las diferentes métricas, de media, en las 10 iteraciones realizadas.

Como podemos observar, los resultados obtenidos con respecto a MAE con únicamente inliers mejoran los resultados tanto con respecto a los datos de test completos como a los del modelo base. En las otras métricas también podemos ver este comportamiento, aunque llama la atención el caso de R2. Resulta llamativo cómo en el caso de los datos de test completo su valor es tan bajo mientras que con los inliers el valor es dos veces mayor. Esto podría deberse a que se esté dando una situación similar a la que veíamos en el problema sintético 2 (figura 4.1 a la derecha).

Además de esto, hemos obtenido unos diagramas de cajas de las diferentes variables a partir de los datos de entrenamiento con el fin de observar la distribución de los datos en función de si son inliers o outliers. En la figura 4.11 podemos ver, de izquierda a derecha, la distribución del total de datos de entrenamiento como referencia, la de los clasificados como inliers y la de los clasificados como outliers.

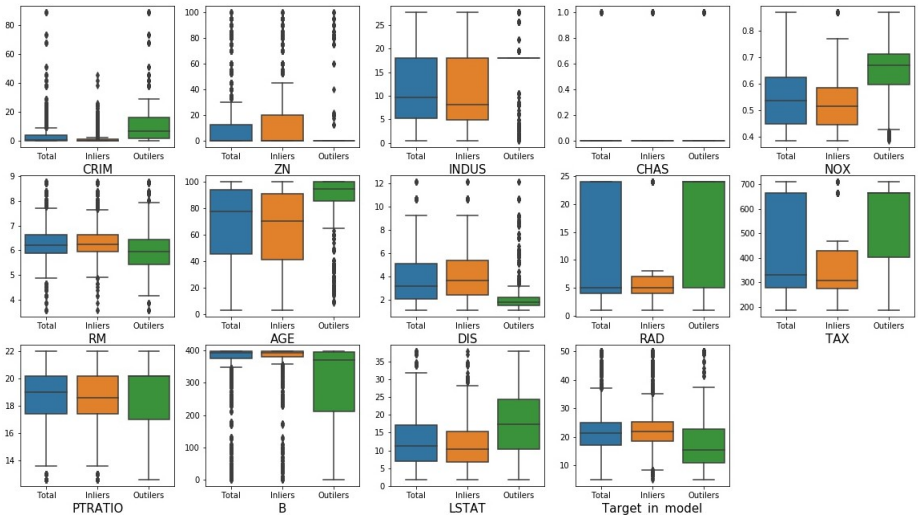

Figura 4.11: Diagrama de cajas obtenido mediante MCD para Boston.

Si nos fijamos en la mediana de las cajas de los outliers, las variables que más destacan son INDUS, NOX, AGE y PTRATIO. En función del rango intercuartílico, las que más llaman la atención son CRIM, ZN, AGE, DIS, RAD, TAX y B.

4.5.3. Detección mediante LOF

Hiperparametrización

Al igual que en el caso anterior, siguiendo la metodología de evaluación del capítulo 4.3.3, vamos a hiperparametrizar este modelo utilizando el método `GridSearchCV` con MAE como métrica de scoring y los diferentes conjuntos de entrenamiento y validación obtenidos mediante `ShuffleSplit` como se ha explicado en 4.3.3 junto con la siguiente rejilla de valores para los diferentes parámetros:

- `contamination`: `[0.05,0.1,0.15,0.2,0.25,0.3,0.4,0.5]`.
- `n_neighbors`: `[5, 9, 17, 33, 65]`$(1+2^k$ con k=2,3,4,5,6).
- `C`: `[0.01,0.1,1,10,100,1000,10000,100000]`.

En esta ocasión, de los 379 patrones utilizados durante el proceso de hiperparametrización del detector de anomalías, se han detectado como outliers un promedio de 65±30 puntos, por lo que el modelo de predicción se ha entrenado con una media de 314±30 puntos.

Resultados en detección de outliers y anomalías

Como en el caso anterior, hemos entrenado una regresión ridge con los datos detectados como inliers en el conjunto de entrenamiento y hemos realizado la evaluación sobre un conjunto de test, tanto manteniéndolo intacto como limpiándolo de outliers. En la tabla 4.13 se muestran los resultados obtenidos para las diferentes métricas, de media, en las 10 iteraciones realizadas.

	Base	MCD		LOF	
Conjunto de test	Completo	Completo	Inliers	Completo	Inliers
MAE	3.458	4.080	2.545	3.590	2.701
MSE	24.779	50.035	11.212	31.743	13.865
R2	0.711	0.406	0.830	0.626	0.799

Tabla 4.13: Resultados en test con LOF en Boston.

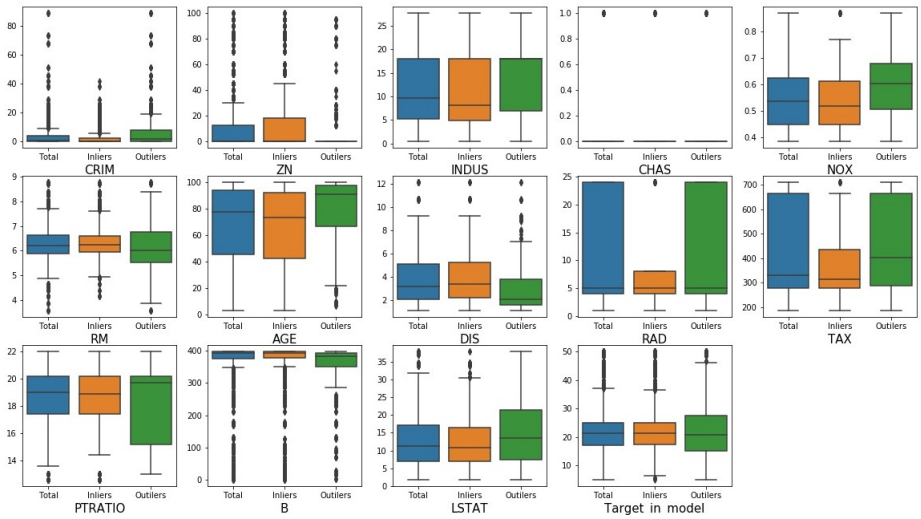

Figura 4.12: Diagrama de cajas obtenido mediante LOF en Boston.

En este caso, el valor tanto de MAE como del resto de métricas vuelven a mejorar los resultados con los datos de test completos y los del modelo base. Aunque estos resultados son ligeramente peores que los obtenidos en el modelo anterior ya no vemos esa gran diferencia en R2 entre los datos completos y los de test, por lo que estos serían más fiables.

De nuevo, hemos obtenido unos diagramas de cajas de las diferentes variables a partir de los datos de entrenamiento con el fin de observar la distribución de los datos en función de si son inliers o outliers. En la figura 4.12 podemos ver, de izquierda a derecha, la distribución del total de datos de entrenamiento como referencia, la de los clasificados como inliers y la de los clasificados como outliers.

En términos de la mediana de los outliers, las variables que más destacan son NOX, AGE y TAX. Si nos fijamos en el rango intercuartílico, las variables más llamativas son ZN, RAD y PTRATIO.

4.5.4. Detección mediante Isolation Forest

Hiperparametrización

Al igual que para los métodos anteriores, siguiendo la metodología de evaluación del capítulo 4.3.3, vamos a hiperparametrizar este modelo utilizando el método `GridSearchCV` con MAE como métrica de scoring y los diferentes conjuntos de entrenamiento y validación obtenidos mediante `ShuffleSplit` como se ha explicado en 4.3.3 junto con la siguiente rejilla de valores para los

	Base	MCD		LOF		IF	
Conjunto de test	Completo	Completo	Inliers	Completo	Inliers	Completo	Inliers
MAE	3.458	4.080	2.545	3.590	2.701	3.427	2.777
MSE	24.779	50.035	11.212	31.743	13.865	27.026	14.771
R2	0.711	0.406	0.830	0.626	0.799	0.684	0.731

Tabla 4.14: Resultados en test con Isolation Forest en Boston.

diferentes parámetros:

- `contamination`: `[0.05,0.1,0.15,0.2,0.25,0.3,0.4,0.5]`.
- `n_estimators`: `[10,25,50,75,100,125]`.
- `max_features`: `[0.125,0.25,0.375,0.5,0.625,0.77,0.875,1.0]`.
- `C`: `[0.01,0.1,1,10,100,1000,10000,100000]`.

En este modelo, de los 379 patrones utilizados durante el proceso de hiperparametrización del detector de anomalías, se han detectado como outliers un promedio de 52±24 puntos, por lo que el modelo de predicción se ha entrenado con una media de 327±24 puntos.

Resultados en detección de outliers y anomalías

Como en los casos anteriores, hemos entrenado una regresión ridge con los datos detectados como inliers en el conjunto de entrenamiento y hemos realizado la evaluación sobre un conjunto de test, tanto manteniéndolo intacto como limpiándolo de outliers. En la tabla 4.14 se muestran los resultados obtenidos para las diferentes métricas, de media, en las 10 iteraciones realizadas.

Con este modelo volvemos a obtener en MAE una puntuación mejor con el conjunto de test de inliers que con el completo. Ademas, esta puntuación también es mejor que la del modelo base, aunque no mejor que las obtenidas anteriormente. Con respecto a las otras métricas sucede lo mismo, son mejores las de solo inliers que las de los datos completos y el modelo base, pero no mejor que las de los modelos anteriores.

De nuevo, hemos obtenido unos diagramas de cajas de las diferentes variables a partir de los datos de entrenamiento con el fin de observar la distribución de los datos en función de si son inliers o outliers. En la figura 4.13 podemos ver, de izquierda a derecha, la distribución del total de datos de entrenamiento como referencia, la de los clasificados como inliers y la de los clasificados como outliers.

Con isolation forest, las variables que más destacan con respecto a los valores de la mediana de outliers e inliers son INDUS, NOX, TAX y PTRATIO, aunque no presentan grandes diferencias. En relación con los rangos intercuartílicos, las variables que mayores diferencias presentan son ZN, RM, PTRATIO y la variable Target.

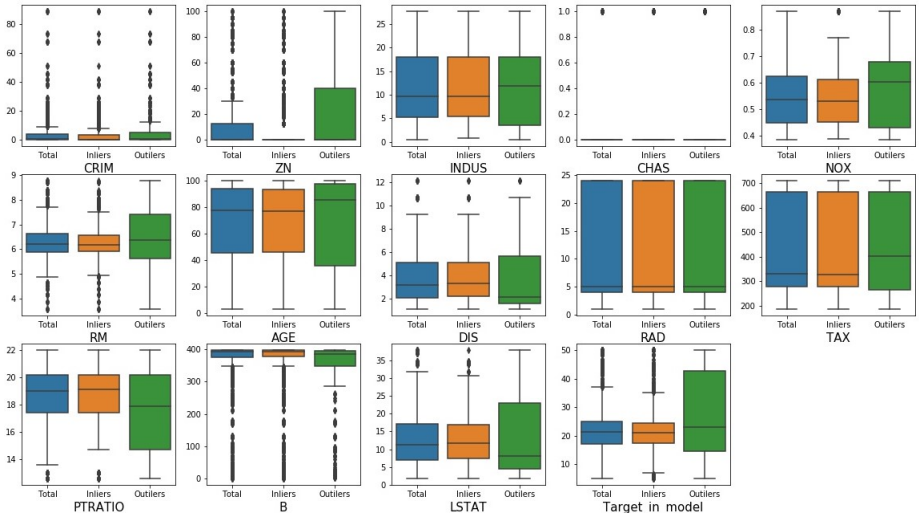

Figura 4.13: Diagrama de cajas obtenido mediante Isolation Forest en Boston.

4.5.5. Detección mediante One-Class SVM

Hiperparametrización

Al igual que para los métodos anteriores, siguiendo la metodología de evaluación del capítulo 4.3.3, vamos a hiperparametrizar este modelo utilizando el método `GridSearchCV` con MAE como métrica de scoring y los diferentes conjuntos de entrenamiento y validación obtenidos mediante `ShuffleSplit` como se ha explicado en 4.3.3 junto con la siguiente rejilla de valores para los diferentes parámetros:

- `nu`: `[0.05,0.1,0.15,0.2,0.25,0.3,0.4,0.5]`.
- `gamma`: `[0.02,0.24,0.46,0.68,0.90,1.12,1.34,1.56,1.78,2.00]`.
- `C`: `[0.01,0.1,1,10,100,1000,10000,100000]`.

En este modelo, de los 379 patrones utilizados durante el proceso de hiperparametrización del detector de anomalías, se han detectado como outliers un promedio de 47±20 puntos, por lo que el modelo de predicción se ha entrenado con una media de 332±20 puntos.

Resultados en detección de outliers y anomalías

Como en los casos anteriores, hemos entrenado una regresión ridge con los datos detectados como inliers en el conjunto de entrenamiento y hemos realizado la evaluación sobre un conjunto de test, tanto manteniéndolo intacto como limpiándolo de outliers. En la tabla 4.15 se muestran los resultados obtenidos para las diferentes métricas, de media, en las 10 iteraciones realizadas.

En esta ocasión, volvemos a observar que el valor de MAE con los inliers es mejor que con los datos completos y que el modelo base. Con las otras métricas también vemos que los resultados son mejores solo con inliers que con los datos completos y que el modelo base. Al igual que pasaba con el modelo anterior, aunque los resultados son buenos no estamos obteniendo una mejora con respecto a los primeros modelos.

	Base	MCD		LOF		IF		One-class	
Conjunto de test	Completo	Completo	Inliers	Completo	Inliers	Completo	Inliers	Completo	Inliers
MAE	3.458	4.080	2.545	3.590	2.701	3.427	2.777	3.436	2.880
MSE	24.779	50.035	11.212	31.743	13.865	27.026	14.771	26.733	15.897
R2	0.711	0.406	0.830	0.626	0.799	0.684	0.731	0.688	0.754

Tabla 4.15: Resultados en test con One-class SVM en Boston.

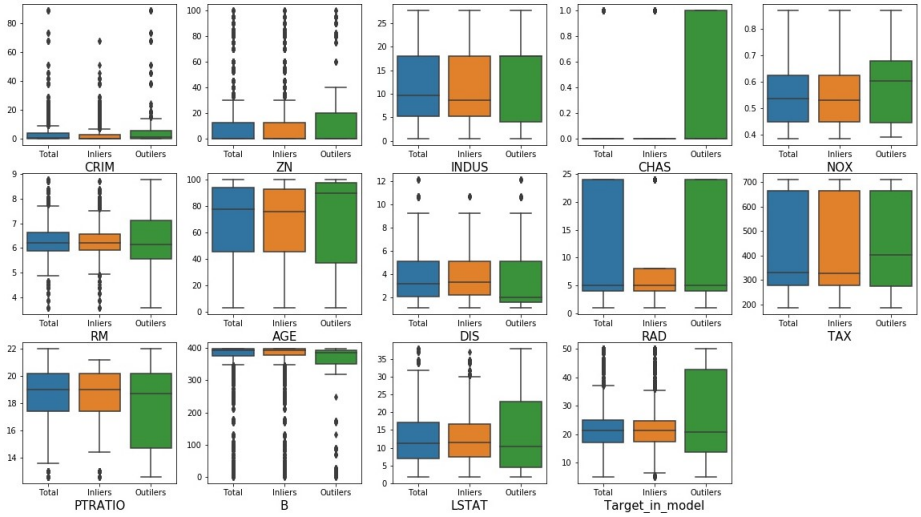

Figura 4.14: Diagrama de cajas obtenido mediante One-Class SVM en Boston.

De nuevo, hemos obtenido unos diagramas de cajas de las diferentes variables a partir de los datos de entrenamiento con el fin de observar la distribución de los datos en función de si son inliers o outliers. En la figura 4.14 podemos ver, de izquierda a derecha, la distribución del total de datos de entrenamiento como referencia, la de los clasificados como inliers y la de los clasificados como outliers.

En este caso, las variables cuyas medianas presentan más diferencias entre anomalías e inliers son INDUS, NOX, AGE, DIS y TAX. Si nos fijamos en los rangos intercuartílicos, las variables que más llaman la atención son CHAS, RAD, PTRATIO y la variable Target.

4.5.6. Detección mediante votación

Una vez analizados los outliers detectados por cada uno de los métodos por separado, vamos a analizar estas detecciones de manera conjunta. Para ello, en primer lugar vamos a comprobar las coincidencia de puntos detectados como outliers entre los diferentes métodos. En la tabla 4.16 podemos la cantidad de coincidencias que hay de media entre los diferentes métodos.

Después de esto, nos puede interesar simular un sistema de detección de outliers por votación. De esta manera, cada método decidirá si un punto es un outlier o no, para finalmente seleccionar como tal los que han sido clasificados como anomalías por varios métodos, como se explica en la sección 4.3.3. En la tabla 4.17 podemos ver por cuántos modelos han sido detectados como outliers, de media, los diferentes patrones del conjunto de entrenamiento del conjunto de datos.

Ahora, podemos crear un nuevo conjunto de datos en el que solo consideramos como outliers

	MCD	LOF	iForest	One-class
MCD	66.500	28.800	23.300	23.600
LOF	28.800	64.600	22.500	23.300
iForest	23.300	22.500	51.300	25.700
One-class	23.600	23.300	25.700	46.100

Tabla 4.16: Coincidencia de outliers entre métodos en Boston.

Coincidencias	0	1	2	3	4
Puntos	249.8	68.2	32.3	19.1	9.6

Tabla 4.17: Número medio de veces que un punto ha sido detectado como outlier en Boston.

	Base	MCD		LOF		IF		One-class		Votación	
Conjunto de test	Completo	Completo	Inliers	Completo	Inliers	Completo	Inliers	Completo	Inliers	Completo	Inliers
MAE	3.458	4.080	2.545	3.590	2.701	3.427	2.777	3.436	2.880	3.631	2.547
MSE	24.779	50.035	11.212	31.743	13.865	27.026	14.771	26.733	15.897	34.410	11.782
R2	0.711	0.406	0.830	0.626	0.799	0.684	0.731	0.688	0.754	0.594	0.796

Tabla 4.18: Resultados en test con votación en Boston.

aquellos puntos que han sido detectados por dos o más métodos, mientras que, los que solo lo han sido por uno los vamos a considerar como inliers.

Como en los casos anteriores, hemos entrenado una regresión ridge con los datos detectados como inliers en el conjunto de entrenamiento y hemos realizado la evaluación sobre un conjunto de test, tanto manteniéndolo intacto como limpiándolo de outliers. En la tabla 4.18 se muestran los resultados obtenidos para las diferentes métricas, de media, en las 10 iteraciones realizadas.

De nuevo volvemos a ver que el valor de MAE con los datos de test de inliers es mejor que en el caso de los datos de test completos y que el modelo base. Además de esto, también mejora o se queda muy cerca de los resultados del resto de métodos. Esto mismo también pasa con el resto de métricas.

Al igual que en anteriores ocasiones, hemos obtenido unos diagramas de cajas de las diferentes variables a partir de los datos de entrenamiento con el fin de observar la distribución de los datos en función de si son inliers o outliers. En la figura 4.15 podemos ver, de izquierda a derecha, la distribución del total de datos de entrenamiento como referencia, la de los clasificados como inliers y la de los clasificados como outliers.

Mediante votación, las medianas de inliers y outliers presentan más diferencias en las variables INDUS, NOX, AGE y TAX. Con respecto a los rangos intercaurtílicos las variables que más llaman la atención son ZN, RAD, TAX, PTRATIO, LTSTAT y la variable Target.

4.5.7. Conclusiones

Como hemos visto, en este problema los resultados son positivos. A excepción del método MCD, todos los métodos en mayor o menor medida mejoran los resultados del modelo base, siendo el de votación el que de forma general mejores estadísticas ofrece.

Además, estos métodos están consiguiendo diferenciar las principales características de los outliers

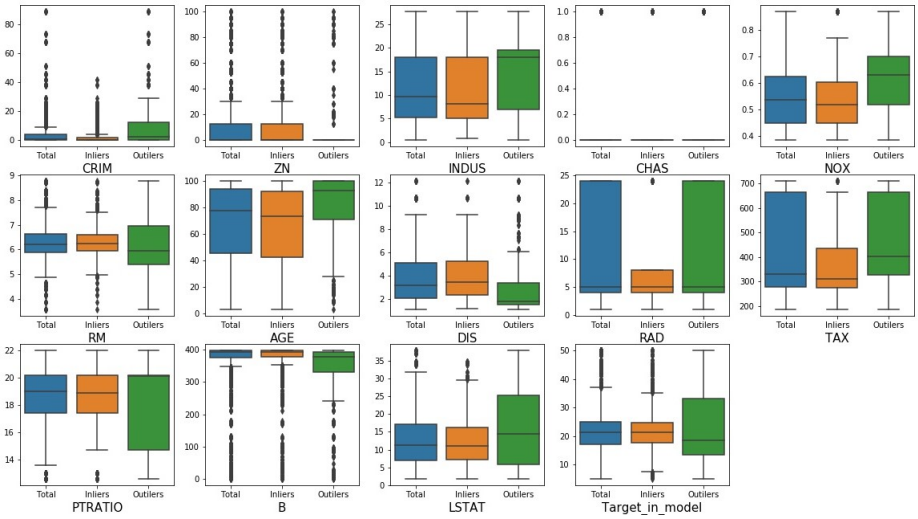

Figura 4.15: Diagrama de cajas obtenido mediante votación en Boston.

de las de los inliers, como hemos podido ver en los diferentes diagramas de cajas. Aunque hay muchas variables y en función del método destacan más unas u otras, si que se puede apreciar que las que destacan en más ocasiones son NOX, INDUS, AGE, TAX, ZN, RAD y PTRATIO. También hay que destacar la inclusión de la variable Target en el modelo, que en varias ocasiones nos permite diferenciar los valores de los inliers y de las anomalías.

4.6. Detección de outliers con datos de predicción eólica en Australia

4.6.1. Descripción del problema y los datos

Para este apartado vamos a utilizar un conjunto de datos de predicción eólica extraídos de la competición GEF de 2014 [18]. Este problema consiste en predecir la energía que se va a producir en un parque eólico en función de diferentes variables, en este caso relacionadas con el viento.

Esta muestra está compuesta por 6576 datos correspondientes a un parque eólico situado en Australia con tomas de datos cada hora desde el 1 de enero de 2012 hasta el 30 de septiembre del mismo año, con las siguientes variables:

- U10: componente U (de oeste a este) del viento a 10 metros en m/s.
- V10: componente V (de sur a norte) del viento a 10 metros en m/s.
- U100: componente U del viento a 100 metros en m/s.
- V100: componente V del viento a 100 metros en m/s.
- v10: velocidad absoluta del viento a 10 metros en m/s.
- v100: velocidad absoluta del viento a 100 metros en m/s.
- Target: energía producida normalizada al rango [0,100].

	count	mean	std	min	25 %	50 %	75 %	max
U10	6576.000	0.926	2.554	-7.494	-1.091	0.780	2.483	11.117
V10	6576.000	-0.248	2.986	-9.994	-2.380	0.011	1.964	9.066
U100	6576.000	1.588	4.249	-10.911	-1.773	1.413	4.642	16.988
V100	6576.000	-0.485	5.126	-15.295	-4.799	0.304	3.615	14.314
v10	6576.000	3.633	1.775	0.115	2.200	3.326	4.792	11.743
v100	6576.000	6.328	2.654	0.076	4.432	6.176	8.004	18.487
Target	6576.000	30.994	29.565	0.000	5.834	21.361	50.155	99.953

Tabla 4.19: Descripción de los datos eólicos.

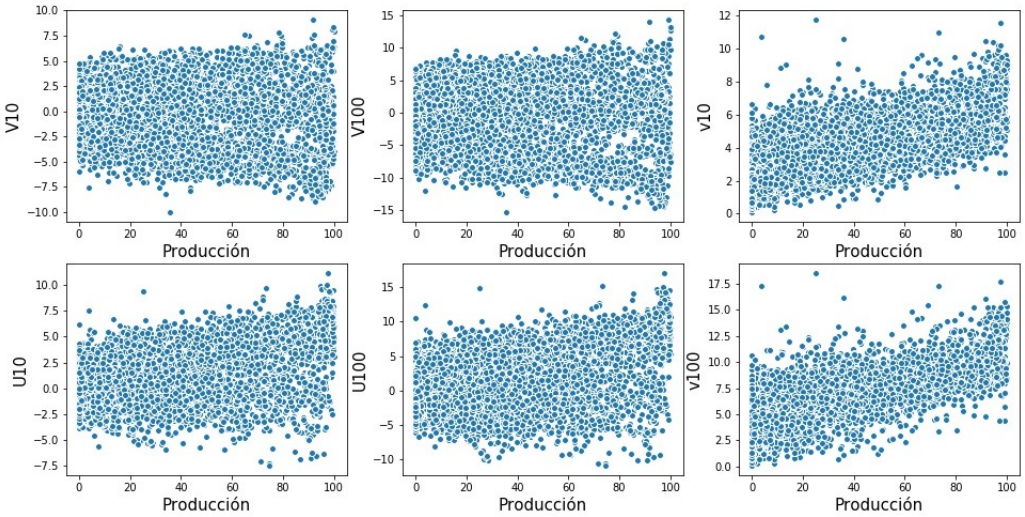

Figura 4.16: Correlación de las distintas variables con la producción de energía eólica en Australia.

En la tabla 4.19 podemos ver una descripción de los datos. En esta tabla podemos ver como en un principio los datos están bastante normalizados aunque sí que podemos observar que los valores máximos y mínimos se alejan bastante del primer y tercer cuartil, lo que nos puede indicar una presencia de outliers.

A la hora de trabajar con este problema hay que tener en cuenta que se trata de un problema relativamente complejo, ya que, como podemos observar en la figura 4.16, no hay mucha correlación entre las distintas variables y la producción asociada a ellas, especialmente en el caso de las componentes U y V del viento, tanto a 10 como a 100 metros.

En esta ocasión, al tratarse de un problema con datos con marca de tiempo, la división en conjuntos de entrenamiento y test vamos a realizarla de diferente manera. Para entrenar los modelos como se detalla en el apartado 4.3.3 vamos a utilizar los datos de los 8 primeros meses y para los test vamos a utilizar los datos del último mes. Debido a que ahora estos conjuntos son estáticos, no tiene sentido realizar las 10 iteraciones en el modelo base, en LOF y en One-class SVM ya que van a ser todas las veces los mismos. En cambio, en el caso de MCD e Isolation Forests, puesto que presentan factores aleatorios (al crear subconjuntos iniciales y al generar los diferentes árboles del bosque respectivamente) sí que se van a realizar las 10 iteraciones para obtener una media de los resultados.

Para establecer un punto de partida, siguiendo la metodología de evaluación explicada en 4.3.3,

	Base
Conjunto de test	Completo
MAE	15.060
MSE	391.449
R2	0.697

Tabla 4.20: Resultados en test sin detección de outliers en Australia.

	Base	MCD	
Conjunto de test	Completo	Completo	Inliers
MAE	15.060	15.016	13.326
MSE	391.449	399.156	291.282
R2	0.697	0.691	0.735

Tabla 4.21: Resultados en test con MCD en Australia.

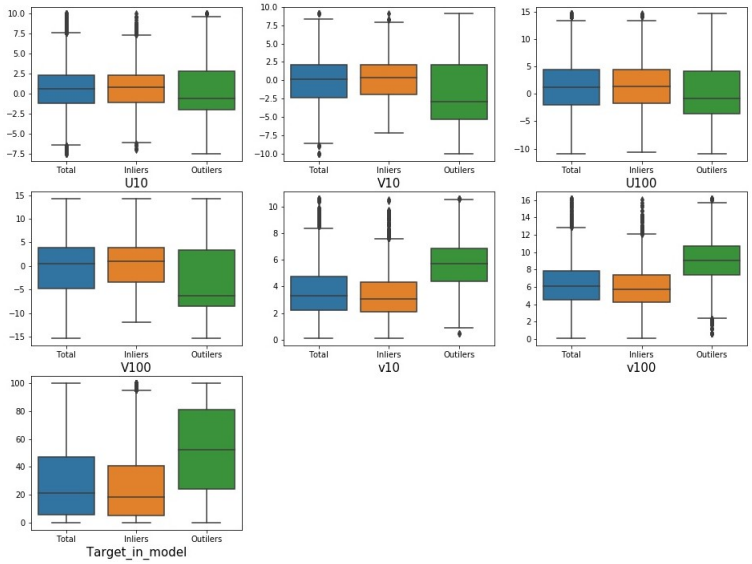

Figura 4.17: Diagrama de cajas obtenido mediante MCD para Australia.

hemos utilizado el método `GridSearchCV` con MAE como métrica de scoring para encontrar el mejor modelo `Ridge` utilizando el conjunto de datos completos, es decir, sin detección de outliers. Para ello hemos probado diferentes valores del parámetro `alpha` (0.01,0.1,1,10,100,1000,10000,100000). Con este modelo, tars las 10 iteraciones, hemos conseguido una puntuación MAE media de 15.471 en entrenamiento. En la tabla 4.20 podemos observar los resultados obtenidos en test.

4.6.2. Detección mediante MCD

Hiperparametrización

Para hiperparametrizar este modelo se ha utilizado el método `GridSearchCV` con MAE como métrica de scoring y los diferentes conjuntos de entrenamiento y validación obtenidos mediante

`ShuffleSplit` como se ha explicado en 4.3.3 junto con la siguiente rejilla de valores para los diferentes parámetros:

- `contamination`: `[0.05,0.1,0.15,0.2,0.25,0.3,0.4,0.5]`.
- `support_fraction`: `[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9]`.
- `alpha`: `[0.01,0.1,1,10,100,1000,10000,100000]`.

De los 5845 patrones utilizados durante el proceso de hiperparametrización del detector de anomalías, se han detectado como outliers un promedio de 790±134 puntos, por lo que el modelo de predicción se ha entrenado con una media de 5055±134 puntos.

Resultados en detección de outliers y anomalías

Como se explica en la sección 4.3.3, tras la creación de un conjunto de datos formado únicamente por los puntos detectados como inliers tras la etapa de hiperparametrización, se ha entrenado una regresión ridge con dichos puntos y se ha evaluado su eficiencia mediante el uso de un conjunto de test, tanto manteniéndolo intacto como limpiándolo de outliers. En la tabla 4.21 se muestran los resultados obtenidos para las diferentes métricas, de media, en las 10 iteraciones realizadas.

Si observamos los resultados obtenidos con respecto a MAE, podemos ver que con los datos de test completos no estamos obteniendo una mejora con respecto al modelo base, pero con los datos de test formados por inliers, sí. Este mismo comportamiento lo podemos ver en el resto de métricas.

Además de esto, hemos obtenido unos diagramas de cajas de las diferentes variables a partir de los datos de entrenamiento con el fin de observar la distribución de los datos en función de si son inliers o outliers. En la figura 4.17 podemos ver, de izquierda a derecha, la distribución del total de datos de entrenamiento como referencia, la de los clasificados como inliers y la de los clasificados como outliers.

Si nos fijamos en la mediana de las cajas de los inliers y los outliers, las variables que más diferencias presentan son V10, V100, v10, v100 y la variable Target. En términos del rango intercuartílico las variables que más destacan son v10, v100 y Target.

4.6.3. Detección mediante LOF

Hiperparametrización

Al igual que en el caso anterior, vamos a hiperparametrizar este modelo utilizando el método `GridSearchCV` con MAE como métrica de scoring y los diferentes conjuntos de entrenamiento y validación obtenidos mediante `ShuffleSplit` como se ha explicado en 4.3.3 junto con la siguiente rejilla de valores para los diferentes parámetros:

- `contamination`: `[0.05,0.1,0.15,0.2,0.25,0.3,0.4,0.5]`.
- `n_neighbors`: `[5, 9, 17, 33, 65]`($1+2^k$ con k=2,3,4,5,6).
- `C`: `[0.01,0.1,1,10,100,1000,10000,100000]`.

	Base	MCD		LOF	
Conjunto de test	Completo	Completo	Inliers	Completo	Inliers
MAE	15.060	15.016	13.326	15.210	12.777
MSE	391.449	399.156	291.282	411.284	250.965
R2	0.697	0.691	0.735	0.681	0.780

Tabla 4.22: Resultados en test con LOF en Australia.

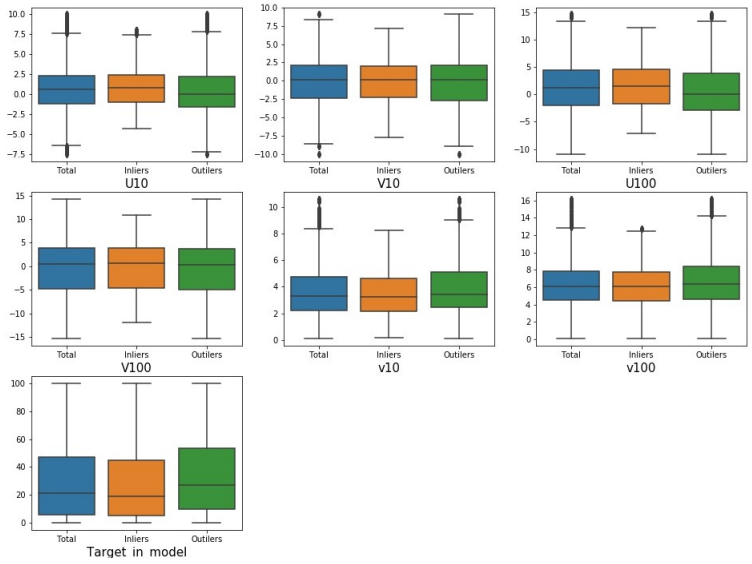

Figura 4.18: Diagrama de cajas obtenido mediante LOF en Australia.

En esta ocasión, de los 5845 patrones utilizados durante el proceso de hiperparametrización del detector de anomalías, se han detectado como outliers 1650 puntos, por lo que el modelo de predicción se ha entrenado con 4195 puntos.

Resultados en detección de outliers y anomalías

Como en el caso anterior, siguiendo la metodología de evaluación del capítulo 4.3.3, hemos entrenado una regresión ridge con los datos detectados como inliers en el conjunto de entrenamiento y hemos realizado la evaluación sobre un conjunto de test, tanto manteniéndolo intacto como limpiándolo de outliers. En la tabla 4.22 se muestran los resultados obtenidos para las diferentes métricas.

Con LOF volvemos a ver cómo los resultados de MAE con los inliers mejoran tanto los del conjunto de test completo, como los del modelo base. Con el resto de métricas también podemos observar una mejora con respecto a las dos situaciones. Además, los resultados de todas las métricas mejoran a los obtenidos con el método de MCD.

De nuevo, hemos obtenido unos diagramas de cajas de las diferentes variables a partir de los datos de entrenamiento con el fin de observar la distribución de los datos en función de si son inliers o outliers. En la figura 4.18 podemos ver, de izquierda a derecha, la distribución del total de datos de entrenamiento como referencia, la de los clasificados como inliers y la de los clasificados como outliers.

	Base	MCD		LOF		IF	
Conjunto de test	Completo	Completo	Inliers	Completo	Inliers	Completo	Inliers
MAE	15.060	15.016	13.326	15.210	12.777	15.662	12.480
MSE	391.449	399.156	291.282	411.284	250.965	435.249	279.795
R2	0.697	0.691	0.735	0.681	0.780	0.663	0.485

Tabla 4.23: Resultados en test con Isolation Forest en Australia.

En el caso de LOF, no se aprecian grandes diferencias en los valores de las medianas entre outliers e inliers. La única variable que llama la atención es la variable Target. En el caso de los rangos intercuartílicos pasa lo mismo, la variable Target es la que más destaca en este sentido.

4.6.4. Detección mediante Isolation Forest

Hiperparametrización

Al igual que para los métodos anteriores, vamos a hiperparametrizar este modelo utilizando el método `GridSearchCV` con MAE como métrica de scoring y los diferentes conjuntos de entrenamiento y validación obtenidos mediante `ShuffleSplit` como se ha explicado en 4.3.3 junto con la siguiente rejilla de valores para los diferentes parámetros:

- `contamination`: `[0.05,0.1,0.15,0.2,0.25,0.3,0.4,0.5]`.
- `n_estimators`: `[10,25,50,75,100,125]`.
- `max_features`: `[0.125,0.25,0.375,0.5,0.625,0.77,0.875,1.0]`.
- `C`: `[0.01,0.1,1,10,100,1000,10000,100000]`.

En este modelo, de los 5845 patrones utilizados durante el proceso de hiperparametrización del detector de anomalías, se han detectado como outliers un promedio de 1988±664 puntos, por lo que el modelo de predicción se ha entrenado con una media de 3857±664 puntos. Como pasaba en el problema de los indios Pima, este modelo está obteniendo una desviación típica bastante alta, lo que puede indicar que no es muy estable.

Resultados en detección de outliers y anomalías

Como en los casos anteriores, siguiendo la metodología de evaluación del capítulo 4.3.3, hemos entrenado una regresión ridge con los datos detectados como inliers en el conjunto de entrenamiento y hemos realizado la evaluación sobre un conjunto de test, tanto manteniéndolo intacto como limpiándolo de outliers. En la tabla 4.23 se muestran los resultados obtenidos para las diferentes métricas, de media, en las 10 iteraciones realizadas.

En este caso, los resultados obtenidos en MAE con los inliers vuelven a ser mejores que los obtenidos con los datos completos y que con el modelo base, pero no mejoran los del modelo anterior. Esto mismo sucede con la métrica MSE. En cambio, en R2 podemos observar cómo en el caso de los inliers la puntuación es muy baja. Esto puede estar sucediendo por la misma razón por la que el número de anomalías detectadas durante el entrenamiento varía tanto.

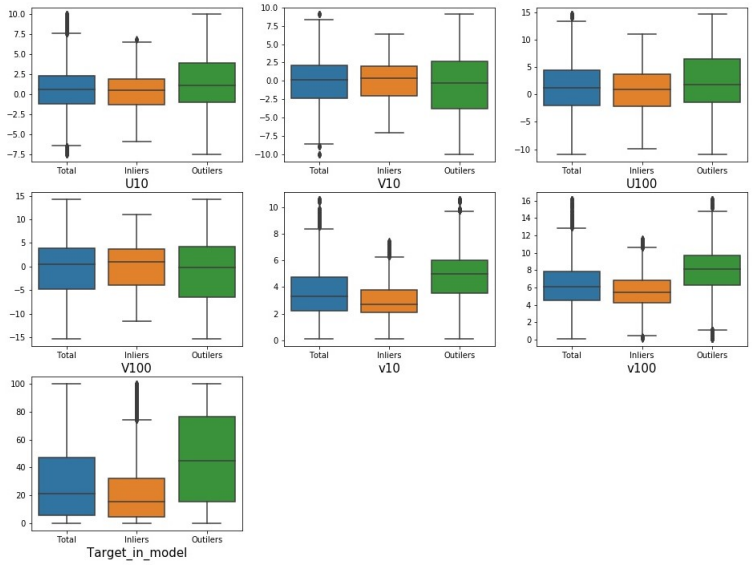

Figura 4.19: Diagrama de cajas obtenido mediante Isolation Forest en Australia.

De nuevo, hemos obtenido unos diagramas de cajas de las diferentes variables a partir de los datos de entrenamiento con el fin de observar la distribución de los datos en función de si son inliers o outliers. En la figura 4.19 podemos ver, de izquierda a derecha, la distribución del total de datos de entrenamiento como referencia, la de los clasificados como inliers y la de los clasificados como outliers.

Con respecto a la diferencia entre outliers e inliers de los valores de la mediana de las variables, las que más destacan son v10, v100 y la variable Target. Si nos centramos en el rango intercuartílico las que más llaman la atención vuelven a ser estas tres, v10, v100 y Target.

4.6.5. Detección mediante One-Class SVM

Hiperparametrización

Al igual que para los métodos anteriores, siguiendo la metodología de evaluación del capítulo 4.3.3, vamos a hiperparametrizar este modelo utilizando el método `GridSearchCV` con MAE como métrica de scoring y los diferentes conjuntos de entrenamiento y validación obtenidos mediante `ShuffleSplit` como se ha explicado en 4.3.3 junto con la siguiente rejilla de valores para los diferentes parámetros:

- `nu`: `[0.05,0.1,0.15,0.2,0.25,0.3,0.4,0.5]`.
- `gamma`: `[0.02,0.24,0.46,0.68,0.90,1.12,1.34,1.56,1.78,2.00]`
- `C`: `[0.01,0.1,1,10,100,1000,10000,100000]`.

En este modelo, de los 5845 patrones utilizados durante el proceso de hiperparametrización del detector de anomalías, se han detectado como outliers 586 puntos, por lo que el modelo de predicción se ha entrenado con 5259 puntos.

	Base	MCD		LOF		IF		One-class	
Conjunto de test	Completo	Completo	Inliers	Completo	Inliers	Completo	Inliers	Completo	Inliers
MAE	15.060	15.016	13.326	15.210	12.777	15.662	12.480	14.993	13.644
MSE	391.449	399.156	291.282	411.284	250.965	435.249	279.795	397.313	299.172
R2	0.697	0.691	0.735	0.681	0.780	0.663	0.485	0.692	0.688

Tabla 4.24: Resultados en test con One-class SVM en Australia.

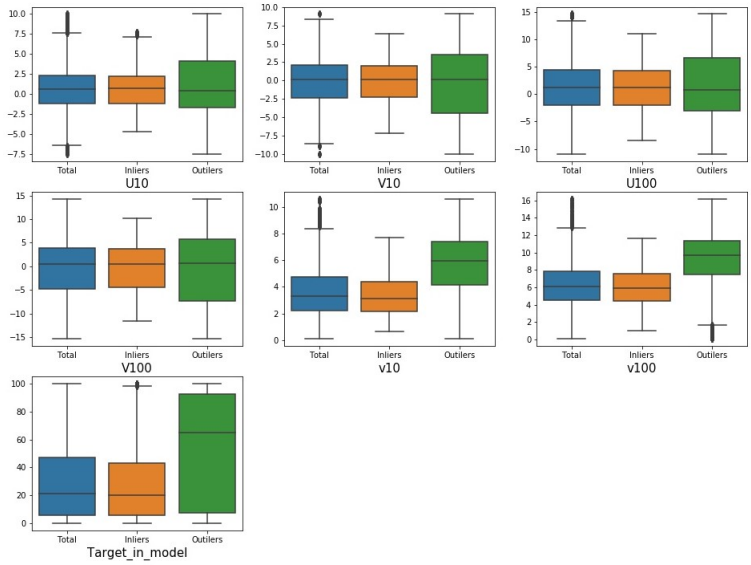

Figura 4.20: Diagrama de cajas obtenido mediante One-Class SVM en Australia.

Resultados en detección de outliers y anomalías

Como en los casos anteriores, hemos entrenado una regresión ridge con los datos detectados como inliers en el conjunto de entrenamiento y hemos realizado la evaluación sobre un conjunto de test, tanto manteniéndolo intacto como limpiándolo de outliers. En la tabla 4.24 se muestran los resultados obtenidos para las diferentes métricas.

Los resultados obtenidos en MAE mejoran los del modelo base tanto con el conjunto de test completo como únicamente con los inliers, siendo este último el mejor de los dos. Con respecto a las otras métricas, los resultados del conjunto completo no mejoran los del modelo base, pero los del conjunto de inliers sí. De nuevo, aunque los resultados son buenos, no están mejorando los ofrecidos por el método LOF.

Una vez más, hemos obtenido unos diagramas de cajas de las diferentes variables a partir de los datos de entrenamiento con el fin de observar la distribución de los datos en función de si son inliers o outliers. En la figura 4.20 podemos ver, de izquierda a derecha, la distribución del total de datos de entrenamiento como referencia, la de los clasificados como inliers y la de los clasificados como outliers.

Si miramos las medianas, las variables que más llaman la atención son v10, v100 y Target. Estas tres también son las que más destacan si nos fijamos en los rangos intercuartílicos, aunque el resto de variables también presentan ciertas diferencias entre outliers e inliers.

	MCD	LOF	iForest	One-class
MCD	789.400	423.800	542.400	358.100
LOF	423.800	1650.000	702.600	388.000
iForest	542.400	702.600	1987.300	494.000
One-class	358.100	388.000	494.000	586.000

Tabla 4.25: Coincidencia de outliers entre métodos en Australia.

Coincidencias	0	1	2	3	4
Puntos	2763.7	1893.5	678.1	275.8	233.9

Tabla 4.26: Número medio de veces que un punto ha sido detectado como outlier en Australia.

	Base	MCD		LOF		IF		One-class		Votación	
Conjunto de test	Completo	Completo	Inliers	Completo	Inliers	Completo	Inliers	Completo	Inliers	Completo	Inliers
MAE	15.060	15.016	13.326	15.210	12.777	15.662	12.480	14.993	13.644	15.028	12.389
MSE	391.449	399.156	291.282	411.284	250.965	435.249	279.795	397.313	299.172	402.715	248.901
R2	0.697	0.691	0.735	0.681	0.780	0.663	0.485	0.692	0.688	0.688	0.718

Tabla 4.27: Resultados en test con votación en Australia.

4.6.6. Detección mediante votación

Una vez analizados los outliers detectados por cada uno de los métodos por separado, vamos a analizar estas detecciones de manera conjunta. Para ello, en primer lugar vamos a comprobar las coincidencia de puntos detectados como outliers entre los diferentes métodos. En la tabla 4.25 podemos ver la cantidad de coincidencias que hay de media entre los diferentes métodos.

Después de esto, nos puede interesar simular un sistema de detección de outliers por votación. De esta manera, cada método decidirá si un punto es un outlier o no, para finalmente seleccionar como tal los que han sido clasificados como anomalías por varios métodos, como se explica en la sección 4.3.3. En la tabla 4.26 podemos ver por cuántos modelos han sido detectados como outliers, de media, los diferentes patrones del conjunto de entrenamiento del conjunto de datos.

Ahora, podemos crear un nuevo conjunto de datos en el que solo consideramos como outliers aquellos puntos que han sido detectados por dos o más métodos, mientras que, los que solo lo han sido por uno los vamos a considerar como inliers.

Como en los casos anteriores, hemos entrenado una regresión ridge con los datos detectados como inliers en el conjunto de entrenamiento y hemos realizado la evaluación sobre un conjunto de test, tanto manteniéndolo intacto como limpiándolo de outliers. En la tabla 4.27 se muestran los resultados obtenidos para las diferentes métricas, de media, en las 10 iteraciones realizadas.

Al igual que sucedía con los métodos anteriores, los resultados de MAE con los datos de test formados por inliers mejoran tanto los resultados de test completos, como los del modelo base. Esto mismo también sucede con el resto de métricas. Además, los resultados ofrecidos por el método de votación con inliers mejoran los de LOF en MAE y los igualan en MSE. Los resultados de R2 son algo peores que los de LOF, aunque puede deberse a la influencia de Isolation Forests.

Como en anteriores ocasiones, hemos obtenido unos diagramas de cajas de las diferentes variables a partir de los datos de entrenamiento con el fin de observar la distribución de los datos en función

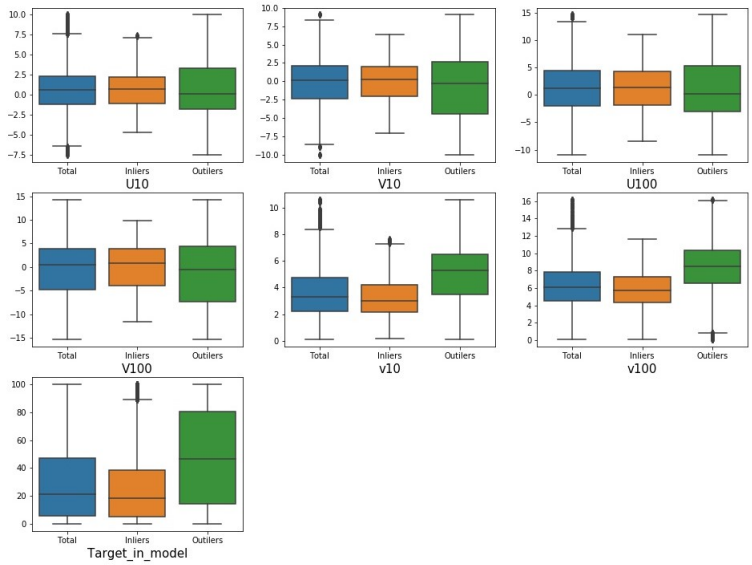

Figura 4.21: Diagrama de cajas obtenido mediante votación en Australia.

de si son inliers o outliers. En la figura 4.21 podemos ver, de izquierda a derecha, la distribución del total de datos de entrenamiento como referencia, la de los clasificados como inliers y la de los clasificados como outliers.

Como sucedía en el método anterior, las variables que más destacan por la diferencia entre la mediana y los rangos intercuartílicos de los outliers y los inliers son v10, v100 y Target.

4.6.7. Conclusiones

A pesar de que se trata de un problema complicado debido a la falta de correlación entre las diferentes variables y la energía producida, podemos observar que hemos conseguido una mejoría en los resultados con respecto al modelo base en la mayoría de los modelos, especialmente con LOF y mediante votación. El único método que ofrece unos resultados menos satisfactorios es Isolation Forest, que, como hemos visto, obtiene una puntuación en R2 especialmente baja.

Al igual que en los problemas anteriores, en esta ocasión también estamos consiguiendo ver diferencias entre la distribución de los datos de las anomalías y la de los inliers. En este caso las variables que más destacan son v10, v100 y la variable Target introducida en el modelo.

5

Conclusiones

Como hemos visto a lo largo de este trabajo, un outlier o anomalía es un punto que no se ajusta a los patrones de comportamiento del resto de puntos de la muestra. Con el fin de identificarlos se han estudiado cuatro métodos:

- Covarianza Robusta.
- Local Outlier Factor.
- Isolation Forests.
- One-class SVM.

Como hemos visto en el capítulo 2 estos cuatro métodos están basados en modelos clásicos de aprendizaje automático. Específicamente hemos estudiado vecinos próximos que es la base de los dos primeros, Random Forest que está directamente relacionado con Isolation Forest y máquinas de vectores de soporte, ya que One-class SVM es una modificación de estas.

Tras el estudio de los diferentes métodos en el capítulo 3 podemos extraer ciertas conclusiones de cada uno de ellos. El método de Covarianza Robusta aunque nos permite encontrar una solución de forma rápida mediante la creación de diversos conjuntos iniciales, si el problema al que nos enfrentamos presenta varios clusters de inliers las soluciones que ofrece son menos satisfactorias que la del resto de métodos. Por otro lado, el método de Local Outlier Factor, aunque se basa en uno de los métodos más sencillos, presenta un gran potencial, ya que permite identificar outliers incluso cuando tenemos varios clusters de inliers de diferentes densidades; aunque para esto debemos ajustar muy bien el número de vecinos a tener en cuenta a la hora de detectar outliers. El método de Isolation Forests, al estar basado en Random Forest, nos permite crear varios árboles con poca correlación entre ellos que nos permite promediar una puntuación que nos indica cómo de anómalo es un punto. Además, como lo que estamos utilizando son árboles, los resultados obtenidos son fácilmente interpretables. Como contrapartida, al igual que en Random Forests, la gran cantidad de parámetros que presenta este modelo hace que la búsqueda de los valores óptimos se más costosa que en el resto de métodos. Por último, One-class SVM presenta

las ventajas de las máquinas de vectores clásicas, como por ejemplo, la posibilidad de utilizar kernels para poder transformar los datos a dimensiones mayores evitando el coste asociado a ello. Esto, añadido a que tiene pocos parámetros, hace que sea un método rápido de hiperparametrizar y que ofrece buenos resultados.

Para comprobar la eficiencia de cada uno de estos métodos, en primer lugar hemos procedido a utilizarlos en dos problemas sintéticos. En estos experimentos hemos podido observar que la detección de anomalías no es algo sencillo, ya que, pequeñas modificaciones en los diferentes parámetros nos llevan a soluciones muy diferentes. Además, podemos observar que la calidad de la solución no viene dada únicamente por el modelo seleccionado sino que también afecta el problema en el que se aplica, como sucede en el caso del compuesto por 4 clusters. En éste podemos observar cómo el método de covarianza robusta no nos ofrece una solución aceptable para ninguna de las combinaciones de valores probadas mientras que para el problema sintético con 2 clusters sí que lo hacía. Estas razones nos llevan a pensar que el proceso de detección de anomalías precisa de una etapa de supervisión en la que se debe analizar si los resultados obtenidos son coherentes con los datos que estamos tratando.

Después de esto, hemos utilizado los diferentes métodos en conjuntos de datos reales, tanto de clasificación como de regresión. La primera situación a la que nos hemos tenido que enfrentar es la escasez de problemas cuyos datos están bien etiquetados como outliers e inliers, y la ausencia de una función score que permita evaluar los resultados de predicción. Para solucionar esto, se ha propuesto un nuevo estimador que nos permite concatenar un método de detección de anomalías con uno de predicción. De esta manera y bajo la idea de que con un conjunto libre de outliers las predicciones realizadas serán más fiables, podemos aplicar la detección de anomalías a cualquier problema.

De esta manera, hemos utilizado el estimador propuesto con los diferentes métodos estudiados para detección de outilers junto con una regresión logística en el caso de problemas de clasificación y un modelo Ridge en el caso de problemas de regresión. Haciendo esto hemos podido comprobar como, en mayor o menor medida, estamos consiguiendo una mejora en la predicción, especialmente cuando aplicamos los modelos de Local Outlier Factor, One-class SVM y el sistema de votación. También hemos podido ver mediante la representación en diagramas de cajas que los diferentes métodos son capaces de diferenciar las principales características de outliers e inliers. De todas maneras, hemos observado que, al igual que sucedía en los problemas sintéticos, los métodos funcionan mejor o peor según el problema en el que se apliquen, por lo que sigue existiendo la necesidad de supervisar los resultados obtenidos.

Con el fin de seguir mejorando los resultados sería interesante probar en el estimador propuesto otros métodos de predicción más potentes como pueden ser SVM's, Random Forests o redes neuronales. Otro aspecto que podría mejorar los resultados es la elección de los métodos utilizados en la votación; de esta manera nos evitaríamos las influencias negativas de aquellos que no son apropiados para ciertos problemas. Finalmente, puesto que la detección de outliers es un campo en auge y se siguen publicando nuevos métodos, como los basados en redes generativas antagónicas (Generative adversarial Networks) [13], los basados en aprendizaje active generativo adverso (Generative Adversarial active learning) [19], los basados en detección de outliers mediante potenciación del gradiente extremo (Extreme Gradient Boosting Outlier Detection) [20] o los basados en la combinación selectiva local en conjuntos paralelos atípicos (Locally Selective Combination in Parallel Outlier Ensembles) [21], también podrían estudiarse y aplicarse estos métodos en el estimador propuesto.

Bibliografía

[1] D. M. Hawkins. *Identification of Outliers*. Springer Netherlands, 1980.

[2] scikit learn developers. Novelty and outlier detection. `https://scikit-learn.org/stable/modules/outlier_detection.html`, 2019.

[3] Peter J. Rousseeuw and Katrien Van Driessen. A fast algorithm for the minimum covariance determinant estimator. *Technometrics*, 41(3):212–223, 1999.

[4] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. LOF: identifying density-based local outliers. *ACM SIGMOD Record*, 29(2):93–104, 2000.

[5] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *2008 Eighth IEEE International Conference on Data Mining*. IEEE, 2008.

[6] Bernhard Schölkopf, John C. Platt, John Shawe-Taylor, Alex J. Smola, and Robert C. Williamson. Estimating the support of a high-dimensional distribution. *Neural Computation*, 13(7):1443–1471, 2001.

[7] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer New York, 2009.

[8] Alberto Torres-Barrán, Álvaro Alonso, and José R. Dorronsoro. Regression tree ensembles for wind energy and solar radiation prediction. *Neurocomputing*, 326-327:151 – 160, 2019.

[9] scikit learn developers. Random forest classifier. `https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html`, 2019.

[10] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer-Verlag New York Inc., 2006.

[11] Thomas M. Cover. Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE Transactions on Electronic Computers*, EC-14(3):326–334, 1965.

[12] Yue Zhao, Zain Nasrullah, and Zheng Li. Pyod: A python toolbox for scalable outlier detection. *Journal of Machine Learning Research (JMLR)*, 2019.

[13] Federico Di Mattia, Paolo Galeone, Michele De Simoni, and Emanuele Ghelfi. A survey on gans for anomaly detection. *CoRR*, 2019.

[14] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation-based anomaly detection. *ACM Transactions on Knowledge Discovery from Data*, 6(1):1–39, 2012.

[15] Shebuti Rayana. ODDS library. `http://odds.cs.stonybrook.edu`, 2016.

[16] María Cristina Tarrés, Nora Moscoloni, Hugo Navone, and Alberto D'ottavio. Análisis multidimensional de una base de datos de mujeres pima. *BIOtecnia*, 18:14–19, 2016.

[17] David Harrison and Daniel L Rubinfeld. Hedonic housing prices and the demand for clean air. *Journal of Environmental Economics and Management*, 5(1):81–102, 1978.

[18] Tao Hong. Gefcom. `http://www.drhongtao.com/gefcom/2014`, 2014.

[19] Yezheng Liu, Zhe Li, Chong Zhou, Yuanchun Jiang, Jianshan Sun, Meng Wang, and Xiangnan He. Generative adversarial active learning for unsupervised outlier detection. *CoRR*, 2018.

[20] Yue Zhao and Maciej K. Hryniewicki. XGBOD: Improving supervised outlier detection with unsupervised representation learning. In *2018 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2018.

[21] Yue Zhao, Zain Nasrullah, Maciej K. Hryniewicki, and Zheng Li. Lscp: Locally selective combination in parallel outlier ensembles. *CoRR*, 2018.

[22] J. Lopez and J. R. Dorronsoro. Simple proof of convergence of the SMO algorithm for different SVM variants. *IEEE Transactions on Neural Networks and Learning Systems*, 23(7):1142–1147, 2012.

[23] Peter J. Rousseeuw and Annick M. Leroy. *Robust Regression and Outlier Detection (Wiley Series in Probability and Statistics)*. Wiley, 1987.

Apéndices

A

Código del estimador implemetado

A continuación se muestra el código de la clase descrita el el apartado 4.3.2.

```
from sklearn.base import BaseEstimator
from sklearn.linear_model import Ridge
from sklearn.svm import OneClassSVM

import pandas as pd
import numpy as np

class OutlierDetector(BaseEstimator):

    def __init__(self, outlier_detector=OneClassSVM(), params_outlier={},
                 model=Ridge(), params_model={}, add_target=False):

        self.outlier_detector = outlier_detector
        self.params_outlier = params_outlier

        self.model = model
        self.params_model = params_model

        self.add_target = add_target

    def fit(self, X, y):

        self.outlier_detector.set_params(**self.params_outlier)
        self.model.set_params(**self.params_model)

        self.outlier_detector.fit(X)

        ind_inliers = np.argwhere(
            self.outlier_detector.predict(X) == 1).flatten()

        # Controlamos si los datos porvienen de un Dataframe o de un array de NumPy
        if isinstance(X, pd.core.frame.DataFrame):
            X_clean = X.iloc[ind_inliers]
            y_clean = y.iloc[ind_inliers]

            # Si entrenamos el detector de outliers con el target
            # debemos quitarlo antes de entrenar el predictor
            # para evitar el overfitting
            if self.add_target:
                X_clean = X_clean[X_clean.columns[:-1]]
        else:
```

```
            X_clean = X[ind_inliers]
            y_clean = y[ind_inliers]
            if self.add_target:
                X_clean = X_clean[:, :-1]

        self.model.fit(X_clean, y_clean)

        self.num_clean_ = X_clean.shape[0]

        return self

    def predict(self, X, y=None):
        try:
            getattr(self, "num_clean_")
        except AttributeError:
            raise RuntimeError(
                "You must train classifer before predicting data!")

        if self.add_target:
            if isinstance(X, pd.core.frame.DataFrame):
                return (self.model.predict(X[X.columns[:-1]]))
            else:
                return (self.model.predict(X[:, :-1]))
        return (self.model.predict(X))

    def score(self, X, y=None):
        return (self.model.score(X, y))

    def decision_function(self, X):
        return self.outlier_detector.decision_function(X)
```